MASTER EN INGÉNIEUR DE GESTION À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

Présentation et mise en pratique d'une méthodologie de développement d'un Data Warehouse

Pisvin, Ethan

Award date:
2019

Awarding institution:
Universite de Namur

Présentation et mise en pratique d'une méthodologie de développement d'un Data Warehouse

Pisvin Ethan

Directeur : Prof. S. Faulkner

Mémoire présenté
en vue de l'obtention du titre de
Maître en Ingénieur de gestion,
orientation Management de l'Information et
finalité spécialisée Data Science

ANNEE ACADEMIQUE 2018-2019

Université de Namur, ASBL
Faculté des Sciences économiques, sociales et de gestion – Département des Sciences de gestion

Rempart de la Vierge 8, B-5000 Namur, Belgique, Tel. +32 [0]81 72 48 41/49 58, Fax +32 [0]81 72 48 40

Avant-propos

Avant toute chose, je tiens à remercier les différentes personnes qui, d'une manière ou d'une autre, m'ont aidé dans la réalisation de ce mémoire.

D'abord, je remercie mon directeur de mémoire, le professeur Stéphane Faulkner, pour ses précieux conseils et son suivi. Je remercie également ma famille, mes amis, et ma petite amie pour leur soutien permanent, sans lequel ce projet n'aurait pas pu être mené à terme. Enfin, je suis également reconnaissant à Pascale Bernard ainsi qu'à Pierre-Yves Outlet, de l'aide qu'ils m'ont apportée dans la relecture et la correction des différents chapitres.

Table des matières

Table des figures

Table des tableaux

Introduction

La Business Intelligence a connu un développement important durant les 30 dernières années. Aujourd'hui, les systèmes OLAP sont largement démocratisés. Les data warehouses et les outils de business intelligence sont utilisés par des organisations de tous types et de toutes tailles, afin de mieux informer les prises de décision des gestionnaires. Dans ce contexte de croissance de la popularité des systèmes reposant sur des data warehouses, il semble pertinent de s'intéresser à la manière de concevoir ces derniers. Cela est d'autant plus intéressant que dans la pratique, il n'est pas rare que des projets de data warehousing échouent en raison d'une mauvaise méthodologie.

L'objet de ce mémoire est donc la présentation d'une méthodologie de développement d'un data warehouse, et la mise en pratique de cette dernière au travers de la réalisation d'une étude de cas. Par la même occasion, ce mémoire vise à mettre en évidence l'importance de l'application d'une méthodologie complète, dans le cadre d'un projet de ce type.

La principale source sur laquelle se base ce travail, est l'ouvrage d'Alejandro Vaisman et Esteban Zimányi intitulé "Data Warehouse Systems : Design and Implementation", dans lequel les deux auteurs présentent leur vision de la méthodologie à adopter dans le cadre du développement d'un data warehouse. La méthodologie présentée ici, s'inspire de celle proposée dans cet ouvrage, et suit en grande partie les mêmes étapes. Les notations utilisées pour les différentes phases de l'étude cas, sont également celles proposées par ces deux auteurs. L'ouvrage de Mattéo Golfarelli et Stefano Rizzi, "Data Warehouse Design : Modern Principles and Methodologies", qui propose une méthodologie similaire, est utilisé en complément de cette source principale, au même titre qu'un série d'articles concernant les divers aspects du design d'un data warehouse.

Ce mémoire compte un total de 6 chapitres. Le premier d'entre eux est consacré à la présentation d'une série de généralités concernant les data warehouses et le modèle multidimensionnel, ainsi qu'à l'introduction de l'étude de cas qui est développée dans les chapitres suivants. Chacun des autres chapitres s'intéresse à une des étapes du processus de design. Pour la plupart d'entre eux, la présentation théorique des concepts et modèles, est suivie par leur application dans le cadre de l'étude de cas.

Dans le deuxième chapitre, c'est la phase de design conceptuel qui est étudiée. Après avoir rappelé l'importance de cette phase, elle est discutée sur le plan méthodologique. La

présentation de la notation Multidim, proposée par Vaisman et Zimányi, occupe également une place importante dans ce chapitre, qui se termine par le design conceptuel du data mart de l'étude de cas.

Le chapitre 3 s'intéresse pour sa part, à la phase de design logique. Après une présentation des différentes possibilités qui s'offrent aux designers pour cette phase, à savoir les modèles ROLAP, MOLAP et HOLAP, le modèle ROLAP est approfondi, car il s'agit du modèle choisi pour l'étude de cas. Ce chapitre s'achève par la traduction du schéma conceptuel obtenu à la fin du chapitre précédent, en schéma logique.

Le chapitre 4 consiste en une présentation théorique de la phase de design physique. Dans ce chapitre, les trois grands moyens d'améliorer les performances d'un data warehouse, à savoir les vues matérialisées, les index, et le partitionnement, sont abordés à tour de rôle.

Le cinquième chapitre traite de l'étape d'ETL. Les premières sections sont utilisées afin de définir le concept de processus ETL, de présenter ses différentes étapes, et d'étudier le problème de la modélisation de ce type de processus. Après cela, les dernières sections sont consacrées à la modélisation conceptuelle du processus ETL de l'étude de cas, ainsi qu'à sa mise en place.

Enfin, le sixième et dernier chapitre, a uniquement pour objectif la finalisation de l'étude de cas. Il se divise en deux parties : la première est dédiée à la définition de la structure multidimensionnelle des données via la création d'un cube SSAS, et la seconde regroupe tout simplement quelques exemples de visualisions, réalisées à partir des données du data mart. Ces visualisations ne sont pas créées afin d'inclure la question du reporting au contenu du mémoire, mais uniquement afin de montrer que le data mart est bien en place et peut être utilisé.

Chapitre 1 : Généralités et présentation de l'étude de cas

Dans ce chapitre, un premier objectif est de présenter une série de généralités concernant le concept de système OLAP, et celui de datawarehouse. Dans un deuxième temps, les grandes idées du modèle multidimensionnel sont introduites. Enfin, la dernière partie de ce premier chapitre est consacrée à l'introduction de l'étude de cas développée tout au long des chapitres suivants.

1.1 Systèmes OLAP et systèmes OLTP

Avant toute chose, il convient d'établir la distinction entre d'une part les systèmes OLTP, et d'autre part les systèmes OLAP. Les premiers aussi appelés bases de données opérationnelles, sont des systèmes pensés, comme leur nom l'indique, à des fins de gestion opérationnelle des activités d'une organisation. Ils contiennent des données détaillées et non historisées, stockées dans des tables hautement normalisées. Ces systèmes sont conçus, de manière à garantir un environnement d'exécution optimal des transactions générées par les opérations quotidiennes au sein d'une organisation (ex : enregistrement des entrées et sorties des stocks). L'exécution de ces transactions, ne requiert généralement l'accès, qu'à un nombre très limité d'enregistrements contenus dans la base de données. Ces systèmes doivent garantir un accès rapide aux données, aussi bien en lecture qu'en écriture, ce qui nécessite de porter une attention particulière aux fonctionnalités de gestion des accès concurrents aux données, ainsi qu'aux mécanismes de garantie de la cohérence des données. Ces derniers tirent avantage de la normalisation des tables, ainsi que des contraintes d'intégrité référentielle, qui réduisent les risques d'apparition d'anomalies lors de la mise à jour des enregistrements dans la base de données (Vaisman et Zimányi, 2014).

Le constat de l'incapacité des systèmes OLTP à satisfaire aux exigences des besoins analytiques des organisations (faibles performances sur requêtes complexes nécessitant le calcul d'agrégations, entrave à l'exécution des transactions de gestion journalière par la monopolisation des ressources du système à des fins d'analyses, non historisation des données, etc.), a mené au développement du paradigme OLAP, pour lequel l'accent est placé sur l'exécution des requêtes analytiques.

En support à ce nouveau paradigme, un nouveau type de systèmes de base de données a vu le jour, les data warehouses. Ils constituent de nouveaux environnements de stockage de données, pensés et optimisés dans l'optique de répondre aux besoins analytiques, qui échappent

aux capacités des systèmes OLTP. Ce changement de priorité nécessite notamment l'introduction d'une certaine forme de redondance dans les données, ce qui va à l'encontre de la normalisation, pourtant essentielle dans le cadre des systèmes opérationnels. On parle de redondance contrôlée (Linden, 2018). Alors que la redondance est supprimée dans les systèmes opérationnels, afin d'optimiser le processus métier et les opérations d'écriture, une certaine forme de redondance est souhaitable dans les data warehouses afin d'en augmenter la performance (Hani Zulkifli Abai, Yahaya et Deraman, 2013).

1.2 Définition du concept de data warehouse

*"Un **data warehouse** est un répertoire de données intégrées obtenues à partir de plusieurs sources dans le but spécifique de réaliser des analyses multidimensionnelles. Plus techniquement, un data warehouse est défini comme une collection de données orientées sujet, intégrées, non-volatiles, et variables dans le temps en support des décisions des gestionnaires."* (Vaisman et Zimányi, 2014)

- **Orientées sujet :** Par opposition aux systèmes et applications orientés fonction, qui sont conçus de manière à remplir une ou plusieurs tâches précises, comme par exemple l'enregistrement des ventes réalisées, les data warehouses répondent aux besoins analytiques de différentes parties d'une organisation, concernant des sujets tels que les ventes, ou encore la gestion du personnel ou des inventaires. Les data warehouses ne sont pas créés afin de remplir une fonction bien définie, mais afin de supporter l'analyse de sujets importants pour les organisations.

- **Intégrées :** Les données contenues au sein d'un data warehouse sont, dans la plupart des cas, extraites de différents systèmes sources. Cela nécessite de surmonter une série d'obstacles, tels que des différences de formats ou de sémantique des données, qui sont autant de barrières à leur intégration au sein d'un même système centralisé, et plus globalement, à une gestion efficace de celles-ci. L'intégration des données au sein d'un seul et unique système, assure l'émergence d'une seule version de la vérité qui garantit la cohérence des analyses.

- **Non-volatiles :** La durabilité des données contenues dans un data warehouse, est assurée en ne permettant ni leur suppression, ni leur modification, une fois chargées dans le data warehouse. En d'autres termes, ce qui entre dans le data warehouse, n'en sort plus. Cette durabilité étend la fenêtre temporelle couverte par les data warehouses. Ceux-ci contiennent généralement des données historisées sur plusieurs années.

- **Variables dans le temps :** Au sein d'un data warehouse, les données sont mises en lien avec une dimension temporelle, qui permet le stockage de différentes valeurs pour une même information à différents points du temps. La présence de cette dimension temporelle est ce qui permet l'historisation des données.

Tableau 1.1 : Comparaison entre les bases de données opérationnelles et les data warehouses (Vaisman et Zimányi, 2014)

	Aspect	Operational databases	Data warehouses
1	User type	Operators, office employees	Managers, executives
2	Usage	Predictable, repetitive	Ad hoc, nonstructured
3	Data content	Current, detailed data	Historical, summarized data
4	Data organization	According to operational needs	According to analysis needs
5	Data structures	Optimized for small transactions	Optimized for complex queries
6	Access frequency	High	From medium to low
7	Access type	Read, insert, update, delete	Read, append only
8	Number of records per access	Few	Many
9	Response time	Short	Can be long
10	Concurrency level	High	Low
11	Lock utilization	Needed	Not needed
12	Update frequency	High	None
13	Data redundancy	Low (normalized tables)	High (denormalized tables)
14	Data modeling	UML, ER model	Multidimensional model

1.3 Les objectifs du data warehouse (Kimball et Ross, 2012)

Dans son ouvrage "The Data Warehouse Toolkit", Ralph Kimball présente une série d'objectifs devant être atteints, et de conditions devant être remplies, dans le cadre d'un projet de data warehousing. Les quelques paragraphes qui suivent présentent ces différents points.

Le data warehouse doit rendre l'information facilement accessible au sein de l'organisation. L'atteinte de cet objectif passe, par des données bien organisées et correctement labélisées, de manière à rendre l'environnement compréhensible par les utilisateurs, et non seulement par les développeurs. Les outils permettant l'accès aux données doivent également être intuitifs, faciles d'utilisation et performants.

Le data warehouse doit présenter l'information d'une organisation de manière cohérente. Beaucoup de soin doit être apporté au rassemblement des données de l'organisation

au sein du data warehouse (nettoyage, transformation, évaluation de qualité, etc.). L'obtention d'informations de qualité, requiert des données cohérentes et complètes en entrée. De plus, l'utilisateur doit avoir à sa disposition une documentation non-ambiguë du contenu du data warehouse.

Le data warehouse doit être adaptatif et résistant au changement. Les organisations et leur environnement étant en constante évolution, le data warehouse doit être en mesure de se plier aux différents changements pouvant survenir (besoins des utilisateurs, réglementations, etc.). Les évolutions doivent pouvoir être implémentées de manière fluide, sans invalidation des données déjà stockées. Autrement dit, le moindre changement ne doit pas nécessiter la remise en cause complète du système et de son architecture.

Le data warehouse doit être une place forte protégeant les informations de l'organisation. Un data warehouse renferme généralement des données et informations sensibles et de valeurs. Il est donc crucial que l'accès à ces données soit minutieusement et efficacement contrôlé.

Le data warehouse doit servir de base à une meilleure prise de décision au sein de l'organisation. Un data warehouse est avant toute autre chose, un système de support à la prise de décision. La valeur ajoutée à une organisation par un tel système, se mesure à la lumière de son impact sur les décisions de l'organisation, et sur leur pertinence. Le data warehouse doit donc contenir les données qui vont permettre d'éclairer les prises de décisions des gestionnaires.

Les membres de l'organisation doivent accepter le data warehouse pour que le projet soit considéré comme un succès. Construire un système performant et élégant est inutile, si personne ne l'utilise. La simplicité d'utilisation, joue ici, un rôle prédominant. Le recours au data warehouse n'étant pas nécessairement obligatoire, si l'utilisation en est compliquée, les utilisateurs auront tendance à choisir de s'en passer.

Tous ces éléments permettent de comprendre à quel point le data warehousing se trouve au croisement de l'IT et du business. La réussite d'un tel projet, nécessite la combinaison de compétences de ces deux champs. Il faut, à la fois, maitriser les compétences métiers nécessaires à l'appréhension du domaine d'application et des demandes des utilisateurs, et les compétences techniques nécessaire à la mise en place et la maintenance du système.

1.4 Le modèle multidimensionnel

Les data warehouses sont basés sur l'idée de stockage des données selon le model multidimensionnel, qui offre une infrastructure propice à l'exploitation des données d'une organisation, à des fins analytiques et de soutien à la prise de décision. Ce model offre notamment une plus grande performance dans l'exécution de requêtes complexes d'agrégation, qui font office de norme dans un contexte analytique, et pour lesquelles les langages tels que le SQL et les bases de données opérationnelles, sont peu adaptés (Vaisman et Zimányi, 2014).

Une métaphore généralement utilisée afin de faciliter l'appréhension de ce modèle, consiste à représenter les données sous la forme d'un cube. Ce concept de "data cube" situé au cœur du modèle multidimensionnel, repose lui-même sur deux éléments principaux : les faits, et les dimensions. Les dimensions constituent les arrêtes du cube, et sont les axes d'analyse des données contenues dans ce dernier. Les faits, pour leur part, sont les cellules qui constituent le cube et auxquelles des valeurs chiffrées sont associées, on parle de mesures.

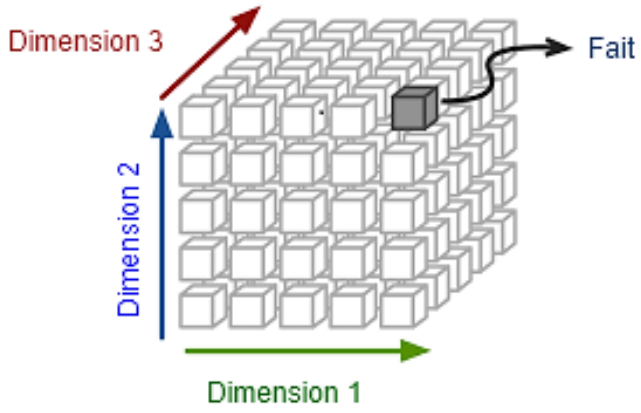


Figure 1.1 : Data Cube

1.4.1 Dimensions

Les dimensions dans une base de données conçue selon le modèle multidimensionnel, sont comme expliqué ci-dessus, autant d'axes d'analyse selon lesquels les utilisateurs peuvent exploiter l'information contenue dans les données. Lorsque qu'un "data cube" présente plus de 3 dimensions, il est question d'un hypercube.

Les dimensions sont choisies parmi les propriétés des faits (cf. section 1.4.3) possédant un domaine de valeurs fini, et peuvent également présenter des attributs descriptifs. Les instances d'une dimension, ou d'un niveau de hiérarchie (cf. section 1.4.2), sont appelées ses membres et sont utilisés pour la sélection et le groupement des données.

1.4.2 Hiérarchies

Les hiérarchies sont des structures logiques prenant la forme d'arbres, qui se greffent sur les dimensions des data warehouses (Golfarelli et Rizzi, 2009). Elles sont constituées d'une série de niveaux, liés entre eux par des dépendances fonctionnelles. Soient deux niveaux reliés l'un à l'autre dans une hiérarchie, le niveau le plus bas des deux est appelé le niveau enfant et l'autre porte le nom de niveau parent. On parle d'une relation parent-enfant.

Elles rendent possible l'analyse des données à différents niveaux de détails. La combinaison des niveaux des hiérarchies sur les différentes dimensions du cube, détermine la granularité des données analysées. Alors que certaines dimensions peuvent présenter plusieurs hiérarchies, d'autres peuvent en être totalement dépourvues.

1.4.3 Faits

Les faits sont des évènements s'étant produits au sein de l'organisation, et dont l'analyse présente une pertinence dans le cadre du processus de prise de décision (Golfarelli et Rizzi, 2009). Ils constituent le sujet de l'analyse. Ils sont décrits par les dimensions, ainsi que par une série d'attributs quantitatifs, appelés les mesures.

Chaque fait, qui pour rappel peut être considéré comme une cellule du cube, est identifié de manière non ambiguë par une combinaison des membres des différentes dimensions du cube. Le cube peut être décrit comme étant dense ou clairsemé, suivant la proportion de faits s'étant effectivement déroulés, ou formulé autrement, la proportion de cellules non-vides dans le cube.

Il existe différents types de faits (Linden, 2018) :

- **Event facts :** Les faits correspondent à des transactions (ex : une vente).
- **"Fact-less" facts :** Les faits sont dépourvus de mesures. La seule information présente est, si oui ou non, un évènement a eu lieu pour une certaine combinaison de membres des dimensions.
- **Snapchot facts :** Les faits capturent le statut de la réalité à un certain point dans le temps (ex : niveau d'inventaire).
- **Cumulative snapshots :** Les faits capturent le statut cumulé de la réalité à un certain point dans le temps (ex : sales year to date).

1.4.4 Mesures

Les mesures représentent les propriétés des faits, que les utilisateurs souhaitent analyser, en vue d'informer leurs prises de décision (Golfarelli et Rizzi, 2009). Une ou plusieurs fonction(s) d'agrégation est/sont associée(s) à chaque mesure. Ces fonctions indiquent la manière dont les valeurs de la mesure sont agrégées/résumées sur les dimensions et le long des hiérarchies du cube.

Il existe différents types de mesures. Ci-dessous sont présentées 2 classifications possibles, faisant référence à la manière dont les mesures sont agrégées (Vaisman et Zimányi, 2014).

Une mesure est dite :

- **Additive** : Lorsqu'elle peut être agrégée de manière sensée sur toutes les dimensions et hiérarchies du data warehouse, en utilisant l'addition.
- **Semi-additive :** Lorsqu'elle peut être agrégée de manière sensée sur certaines des dimensions et hiérarchies du data warehouse en utilisant l'addition, mais pas sur toutes.
- **Non additive :** Lorsqu'elle ne peut être agrégée de manière sensée sur aucune dimension du data warehouse, en utilisant l'addition.

Une mesure est dite :

- **Distributive :** Lorsqu'une agrégation de cette mesure, peut être calculée directement à partir des valeurs de sous-agrégats, sans qu'aucune information supplémentaire ne soit nécessaire à cette fin. C'est le cas des mesures agrégées par le calcul du maximum ou du minimum, par l'addition, ou encore par comptage.
- **Algébrique :** Lorsque le calcul d'une agrégation ne peut pas se faire directement à partir des valeurs de sous-agrégats, et nécessite le recueil d'informations supplémentaires. Un exemple commun de mesure algébrique, est une mesure agrégée grâce à la fonction moyenne. Le calcul de la moyenne de plusieurs sous-agrégats, nécessite la connaissance du nombre d'évènements à partir duquel chacun de ceux-ci a été calculé, à des fins de pondération.
- **Holistique :** Lorsqu'il est impossible de calculer le résultat d'une agrégation, à partir des valeurs de sous-agrégats. Pour ce type de mesures, il est nécessaire de repartir systématiquement du niveau de granularité le plus faible, afin de calculer une

agrégation. Les mesures agrégées en utilisant le mode, la médiane, ou encore le rang, sont des mesures holistiques.

Une propriété particulièrement importante des mesures est connue sous le nom de "**summarizability**". Cette propriété désigne le fait qu'une mesure puisse être correctement agrégée le long des hiérarchies des dimensions, de manière à obtenir des agrégations cohérentes. Pour ce faire, plusieurs conditions doivent être remplies (Vaisman et Zimányi, 2014) :

- **Disjointness of instances :** Cette première condition est remplie lorsque, le long d'une hiérarchie, l'agrégation des membres du niveau enfant vers le niveau parent, résulte systématiquement en la création d'ensembles disjoints. En d'autres termes, chaque membre du niveau enfant doit avoir au plus un parent.
- **Completeness :** Cette condition désigne le fait que tous les membres du niveau enfant, doivent être liés à un parent dans le niveau supérieur.
- **Correctness :** Cette condition désigne l'absence d'erreur lors de l'utilisation des fonctions d'agrégation.

1.5 Introduction de l'étude de cas

1.5.1 Mise en contexte

L'étude de cas qui est développée tout au long de ce mémoire, se base sur la version 2012 de la base de données Adventureworks, mise à disposition par Microsoft. Adventureworks est une société fictive, active dans la production et la commercialisation de vélos, de pièces détachées, ainsi que d'accessoires liés à la pratique du cyclisme. L'objectif de l'étude de cas est la création d'un data mart pour le département des ventes de la société Adventureworks.

1.5.2 Méthodologie

L'approche suivie correspond à une approche "bottom-up", qui consiste à construire, non pas directement un data warehouse complet couvrant l'ensemble des besoins analytiques des différents départements de l'organisation, mais plusieurs data marts qui sont ensuite intégrés sur le plan logique, via une série de dimensions conformes ("Data Marts Bus architecture" proposée par Ralph Kimball). Ces dimensions dites conformes, sont des dimensions communes aux différents data marts, et qui conservent la même signification pour chacun d'eux.

Cette approche est notamment préférée à l'architecture "Hub and Spoke" proposée par Bill Inmon, qui préconise la conception d'un data warehouse central pour l'ensemble de l'organisation, et sur base duquel des data marts sont ensuite constitués (approche "top-down").

Plusieurs facteurs entrent généralement en ligne de compte dans le choix d'une approche plutôt qu'une autre. On peut, entre autres, citer les compétences de l'équipe de développement en charge du projet, l'ampleur du data warehouse à mettre en place, le niveau de motivation des utilisateurs, ou encore le soutien financier apporté à la réalisation des ambitions du projet (Vaisman et Zimányi, 2014). L'objectif étant ici la création d'un seul data mart, et non l'implémentation d'un data warehouse qui couvrirait l'ensemble de l'entreprise, l'approche "bottom-up" s'impose d'elle-même.

De manière générale, l'architecture "Data Marts Bus" présente une série d'avantages. Elle tend à être plus simple à mettre en place, plus rapide à implémenter, et moins couteuse ((Linden, 2018) ; (Ariyachandra et Watson, 2006)). Toutefois, l'évolution des différentes manières de concevoir un système OLAP tend à amoindrir les différences qui les séparent (Ariyachandra et Watson, 2006).

Un data warehouse pouvant être vu comme étant, ni plus ni moins, qu'un type de bases de données dédié à l'analyse et la prise de décision (Vaisman et Zimányi, 2014), la méthodologie utilisée pour le développement de l'étude de cas, suit les étapes traditionnelles du développement de tout système de base de données (cf figure 1.2). À ces étapes sont également ajoutées celle du design et de la mise en place du processus ETL, et celle de la définition de la structure multidimensionnelle des données.

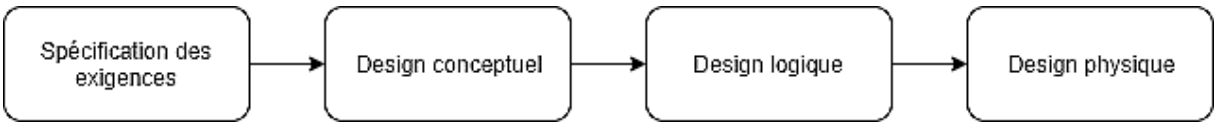


Figure 1.2 : Phases du développement d'un système de base de données

Les technologies et outils qui sont utilisés pour les différentes étapes du processus, sont ceux de la suite Microsoft. Le data warehouse est créé sur SQL Server, le processus ETL est réalisé avec SQL Server Integration Services (SSIS), et la structure multidimensionnelle des données est définie grâce à SQL Server Analysis Services (SSAS). Ce dernier outil est donc le moteur multidimensionnel, qui permet la formulation de requêtes sur le data mart.

1.5.3 Les données à disposition

La totalité des données à disposition est fournie par l'ERP de la société. Ce dernier constitue donc l'unique système source, ce qui facilite grandement la tâche d'intégration des données et le processus ETL. Il n'est, en effet, pas nécessaire de passer par une étape de réconciliation des schémas des systèmes sources.

La tâche se limitant à la création du data mart destiné au département des ventes, seul un sous-ensemble des tables de la base de données source, contient des informations pertinentes. Il s'agit principalement des tables reprises dans la rubrique "Sales" du schéma de données (cf. figure 1.3 ou Annexe 1), ainsi que de certaines tables des rubriques "Person", "HumanResources" et "Production". Une documentation complète de la base de données est disponible sous forme d'un dictionnaire des données (Sqldatadictionary.com, 2016).

Les données de ventes contenues dans la base de données, s'étendent sur une période allant du juillet 2005 à août 2008.

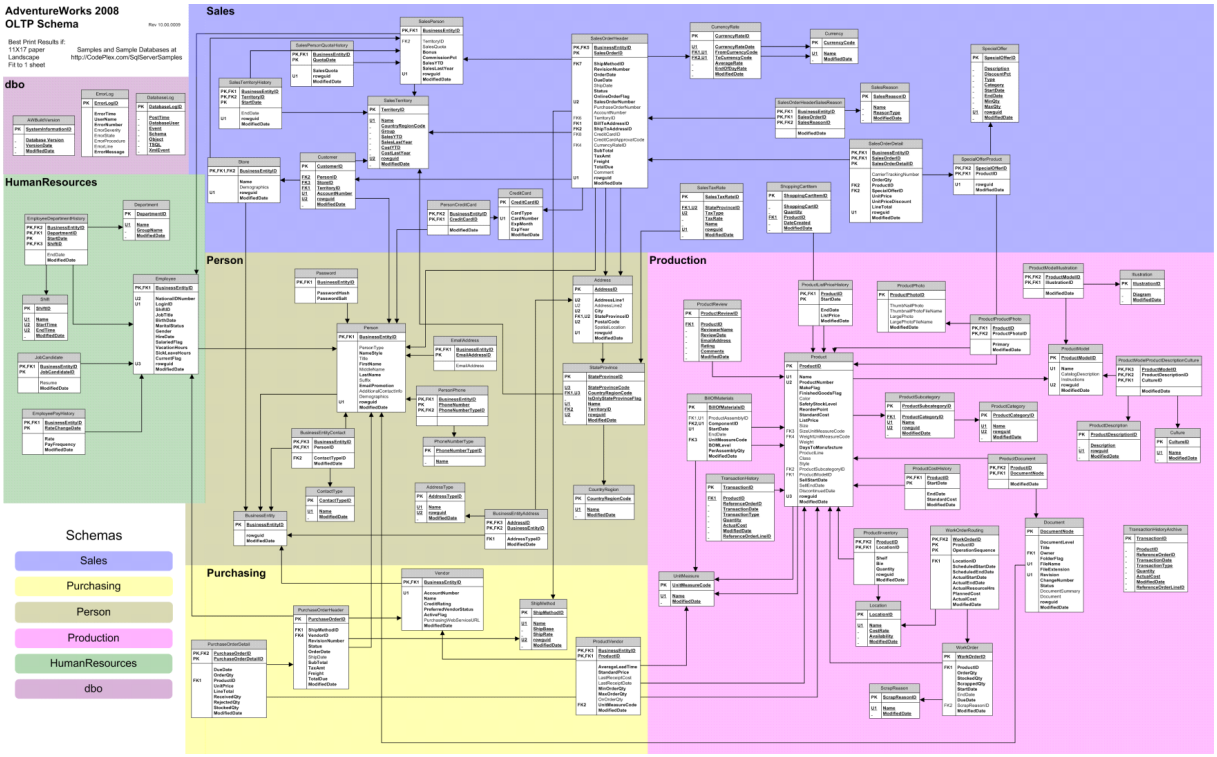


Figure 1.3 : Schéma de données de l'ERP d'Adventureworks

Chapitre 2 : Design Conceptuel

Ce chapitre traite de la phase de design conceptuel. D'abord, une première sous-section est consacrée à la justification de l'utilité de cette étape du processus de design. On s'intéresse ensuite à l'aspect méthodologique de sa mise œuvre, avant de présenter la notation Multidim, proposée par Alejandro Vaisman et Esteban Zimányi. Enfin, après une brève sous-section consacrée à quelques autres notations proposées dans la littérature, la phase de design conceptuel de l'étude de cas est réalisée, au moyen du modèle Multidim.

2.1 La pertinence de l'étape conceptuelle

Il n'existe aucun consensus concernant les différentes phases par lesquelles il faut passer au cours du design d'un data warehouse. Comme énoncé précédemment (cf. section 1.5.2), l'approche qu'il a été choisi de suivre ici, consiste à appliquer au design d'un data warehouse les mêmes phases que celles généralement appliquées à celui d'une base de données relationnelle classique, à savoir : la spécification des exigences, le design conceptuel, le design logique et enfin le design physique. Cette façon de procéder est défendue par une partie des auteurs et praticiens, alors que d'autres ont tendance à ignorer certaines de ces étapes.

L'étape la plus souvent négligée est celle du design conceptuel. Ce phénomène peut s'expliquer, en grande partie, par l'absence d'un modèle bien établi et universellement adopté pour la modélisation conceptuelle de données multidimensionnelles. L'absence, pour les data warehouses, d'un standard équivalent au modèle entités-associations ou encore à UML pour les bases de données relationnelles, a pour conséquence d'inciter nombre de responsables de projets de data warehousing, à passer directement au design logique, sans passer par la case conceptuelle (Vaisman et Zimányi, 2014). Ce manque d'intérêt pour la modélisation conceptuelle, montré par les praticiens comme par les scientifiques, peut également s'expliquer par le fait que ce type de systèmes ait initialement émergé dans un contexte industriel, en réponse à des demandes pratiques de la part d'utilisateurs peu concernés par les questions conceptuelles. L'importance de l'optimisation des performances dans les data warehouses, a aussi poussé les concepteurs à se focaliser sur le design logique et le design physique (Golfarelli, Maio et Rizzi, 1998).

L'utilité de la modélisation conceptuelle dans le contexte des bases de données, est cependant largement admise. Un schéma conceptuel est généralement intuitif, et facilite la communication entre les concepteurs et les futurs utilisateurs du système, ainsi que la validation

des spécifications par ces derniers. Cette simplicité est notamment liée à l'indépendance de ce type de modèles, vis-à-vis de toute plateforme d'implémentation. En plus de rendre le schéma conceptuel par définition plus stable, cette indépendance implique qu'il n'est pas nécessaire de posséder des connaissances particulières concernant la future plateforme d'implémentation, afin de comprendre ce dernier. Selon Vaisman et Zimányi (2014), un schéma conceptuel est une représentation visuelle, concise, et facile à comprendre des exigences des utilisateurs concernant les données, dépourvue de toute considération touchant à l'implémentation du système.

Un autre atout non négligeable des modèles conceptuels, est qu'ils bénéficient généralement d'une plus grande expressivité que leurs homologues de la phase logique. Il est donc possible d'exprimer de manière plus complète les exigences des utilisateurs, et il est également plus simple de les adapter.

Enfin, les schémas créés durant la phase de modélisation conceptuelle, peuvent généralement être traduit en schémas logiques sans difficultés, au moyen de règles de mapping.

2.2 Méthodologie

La première étape du design conceptuel d'un data warehouse, consiste en la spécification des exigences concernant le futur système. Cette étape doit fournir les informations nécessaires à la réalisation du schéma conceptuel initial, qui évolue ensuite au fur et à mesure que les spécifications sont affinées et complétées, et ce peu importe l'approche utilisée.

La formalisation des exigences est d'une importance capitale pour la réussite d'un projet de data warehousing, et impacte l'ensemble des étapes suivantes du processus (Kimball et al., 2008). De nombreux projets ne parviennent, en effet, pas à atteindre leurs objectifs. Certaines des raisons principales qui expliquent ces échecs, touchent à une mauvaise mise en œuvre de l'étape d'analyse et de définition des exigences (Hani Zulkifli Abai, Yahaya et Deraman, 2013). Négliger cette étape, qui tisse le lien entre le business et l'IT, peut donc grandement mettre en péril l'ensemble du processus de design. À l'opposé, une spécification rigoureuse des exigences, et plus globalement une phase de design conceptuel correctement réalisée, permet de s'assurer que le système mis en place sera adapté, et augmente par conséquent les chances de réussite du projet.

Il existe différente manière d'aborder la spécification des exigences. À la fois Vaisman et Zimányi (2014) et Golfarelli et Rizzi (2009) présentent 3 approches pour la spécification des exigences dans le cadre d'un projet de data warehousing :

- **L'approche "analysis-driven" / "requirement-driven" :**

Cette approche consiste, dans un premier temps, à identifier les futurs utilisateurs du système aux différents niveaux hiérarchiques de l'organisation, pour ensuite éliciter leurs exigences. Différentes méthodes, comme par exemple les interviews en profondeur, permettent d'identifier les objectifs de l'organisation ainsi que les exigences des utilisateurs alignées avec ces objectifs, de les opérationnaliser, et ainsi de découvrir les éléments du modèle multidimensionnel pertinents afin de répondre aux besoins d'analyse (faits, mesures, dimensions et hiérarchies). L'idée est donc de parvenir à identifier les données qui doivent être disponibles et exploitables, afin de satisfaire les attentes des futurs utilisateurs.

Cette méthode est caractérisée par la priorité de l'analyse des exigences des utilisateurs, sur l'analyse des sources de données disponibles. Les schémas des systèmes sources n'interviennent généralement qu'une fois le schéma conceptuel initial réalisé, afin de s'assurer de la disponibilité des données et d'établir le mapping avec ces systèmes sources.

En suivant cette approche, on s'assure donc que le schéma initialement obtenu correspond aux attentes des utilisateurs. Il se peut qu'il ne soit toutefois pas supporté par les données effectivement disponibles (Di Tria, Lefons et Tangorra, 2017). Un autre revers de cette approche, est sa dépendance vis-à-vis de l'implication, et de la bonne volonté des futurs utilisateurs à tous les niveaux de l'organisation (Hani Zulkifli Abai, Yahaya et Deraman, 2013).

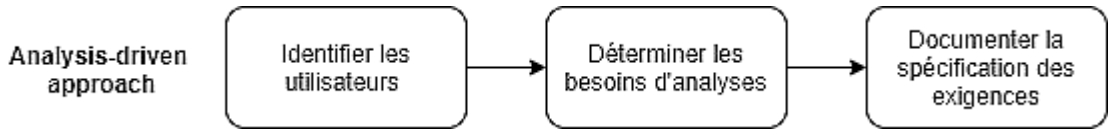


Figure 2.1 : Etapes de l'approche Analysis-driven

- **L'approche "source-driven" / "data-driven" :**

Dans le cadre de cette approche, c'est l'analyse des systèmes sources qui mène à l'élaboration du schéma conceptuel du data warehouse. Une fois les différents systèmes sources identifiés, l'analyse de leur documentation (schémas conceptuels, schémas logiques, etc.) et l'application de procédures de dérivation, permettent l'indentification des

potentiels faits, mesures, dimensions et hiérarchies. Les utilisateurs n'interviennent activement dans le processus que par la suite, afin de vérifier la validité de la structure multidimensionnelle ainsi dérivée ou, afin de la compléter et de l'aiguiller.

La priorité est donc ici donnée, à l'analyse des données disponibles dans les systèmes sources. Les exigences des utilisateurs n'entrent en jeu que dans un second temps, afin de valider le schéma conceptuel initial.

La mise en relation directe de la structure du data warehouse avec celle des systèmes sources, qui caractérise cette approche, permet d'éviter les complications inhérentes à la réalisation ultérieure de ce rapprochement, et d'évaluer d'entrée de jeu le potentiel des données à disposition. Cependant, cette façon de faire présente aussi certains inconvénients. Cette manière de procéder permet d'aboutir à un schéma initial qui garantit la faisabilité du système, mais qui néglige totalement les besoins des utilisateurs (Di Tria, Lefons et Tangorra, 2017). De même, partir des données disponibles dans les systèmes sources permet généralement de faciliter le processus ETL, mais augmente aussi le risque d'un gaspillage de ressources dû à la manipulation de données inutiles, et à la surabondance de données (Hani Zulkifli Abai, Yahaya et Deraman, 2013).

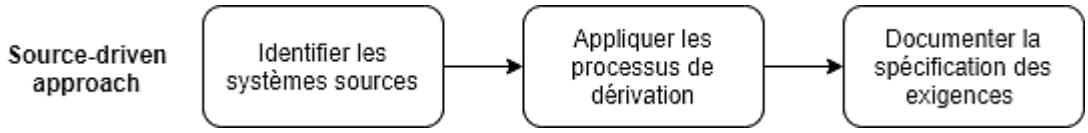


Figure 2.2 : Etapes de l'approche Source-driven

- **L'approche mixte :**

Cette approche est une combinaison des deux précédentes. Le recueil des exigences des utilisateurs, et l'exploration des systèmes sources qui seront amenés à nourrir la data warehouse, sont réalisés en parallèle. Deux schémas conceptuels initiaux sont ainsi obtenus et ensuite confrontés l'un à l'autre, en vue d'obtenir un troisième schéma prenant en considération les deux points de vue. Les deux approches imposent des contraintes l'une sur l'autre, et la confrontation des schémas initiaux peut faire apparaitre la nécessité de révision de leurs exigences par les utilisateurs, ou la nécessité d'analyses supplémentaires des systèmes sources, voire de l'ajout d'autres sources possiblement extérieures à l'organisation. Cette approche permet le développement d'un schéma à la fois en accord avec les exigences des utilisateurs, et cohérent vis-à-vis des données disponibles.

Bien que les méthodologies hybrides ou mixtes impliquent un processus de design globalement plus complexe, les bénéfices qui les accompagnent justifient généralement leur utilisation (Di Tria, Lefons et Tangorra, 2017).

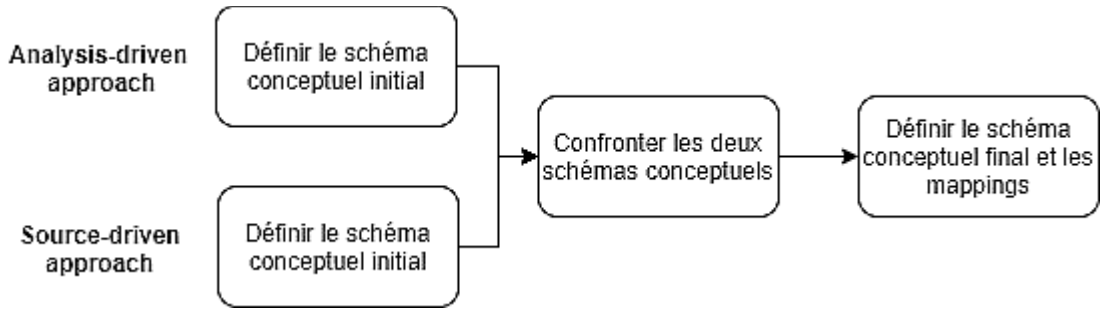


Figure 2.3 : Etapes de l'approche mixte

Comme évoqué brièvement ci-dessus pour chacune des trois méthodes, le choix de l'une ou l'autre approche pour la spécification des exigences et la réalisation du schéma initial, influence la suite de la phase de design conceptuel. Dans le cas de l'approche "requirement-driven", le schéma doit être validé au regard des données effectivement disponibles dans les systèmes sources, ce qui peut pousser les utilisateurs à adapter leurs exigences à ces données effectivement disponibles. Si c'est l'approche "data-driven" qui a été privilégiée, le schéma doit être soumis à la validation des utilisateurs, qui évaluent sa pertinence relativement à leurs besoins d'analyse. Enfin, si c'est une approche mixte qui est adoptée, les deux schémas initiaux doivent être confrontés, de manière qu'ils se contraignent et se complètent l'un l'autre. Il s'agit évidemment d'un processus itératif, pouvant nécessiter de multiples ajustements des exigences des utilisateurs ou des analyses des sources de données, jusqu'à l'obtention du schéma final validé par les utilisateurs et présentant un mapping totalement documenté. C'est ce schéma conceptuel final qui sert de base à la phase de design logique.

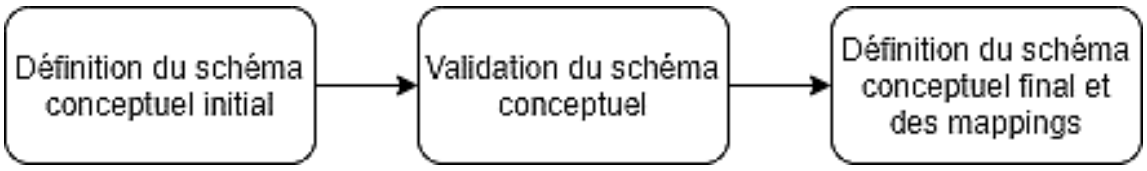


Figure 2.4 : Grandes étapes du design conceptuel d'un data warehouse

En définitive, chacune des approches présente ses avantages et ses inconvénients. Le choix d'une approche au détriment des autres, doit dépendre des caractéristiques de l'organisation : implication du top management et des utilisateurs, niveau de documentation des systèmes sources, compétences de l'équipe chargée du projet, etc. (Hani Zulkifli Abai, Yahaya et Deraman, 2013).

Notons également, que de nombreux autres auteurs proposent dans leurs publications, leurs propres déclinaisons des différentes approches présentées ci-dessus.

2.3 Le langage de modélisation Multidim (Vaisman et Zimányi, 2014)

Multidim est un modèle proposé par Alejandro Vaisman et Esteban Zimányi, pour la schématisation conceptuelle des données multidimensionnelles. Ce modèle est visuellement assez proche du modèle entité-association, qui fait généralement office de référence en ce qui concerne la modélisation conceptuelle des bases de données opérationnelles.

Un schéma Multidim se compose de deux types d'éléments : les faits, et les dimensions. Ils correspondent aux éléments principaux du paradigme multidimensionnel présentés précédemment (cf. section 1.4).

2.3.1 Dimensions

Bien que les dimensions soient présentées par les créateurs du modèle comme l'un de ses deux éléments centraux, Multidim ne comprend pas de symbole directement associé à ce concept. Une dimension est représentée concrètement, soit par un niveau unique, soit par une ou plusieurs hiérarchies.

2.3.2 Niveaux

Un niveau représente un concept du monde réel, analogiquement à une entité dans le cadre du modèle entité-association. Les instances d'un niveau sont appelées les membres de ce niveau. Un niveau possède une série d'attributs qui caractérisent ses membres. Parmi ces attributs se trouve(nt) un ou plusieurs identifiant(s) qui permet(tent) d'identifier sans ambiguïté chacun des membres du niveau. Bien que chaque attribut possède un type associé à son domaine, cette information n'apparait pas sur le schéma (cf. figure 2.5).

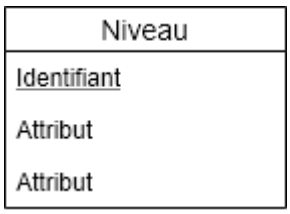


Figure 2.5 : Multidim - Niveau

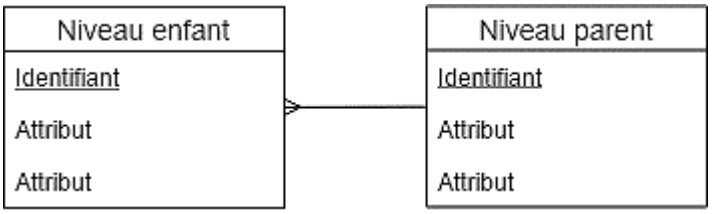


Figure 2.6 : Multidim - Relation parent-enfant

2.3.3 Hiérarchies

Les hiérarchies sont représentées par une série de niveaux liés les uns aux autres par des relations appelées relations parent-enfant (cf. figure 2.6). Elles permettent d'observer les données à différents niveaux de granularité. Le nom d'une hiérarchie, aussi appelé critère d'analyse, est représenté sur le schéma dans une case arrondie, juxtaposée au niveau le plus bas de la hiérarchie en question (cf. figure 2.7). Comme abordé précédemment, une même dimension peut présenter plusieurs hiérarchies, chacune représentant un critère d'analyse différent. Ce type de situation est clairement indiqué sur un schéma Multidim par la présence du nom de chacune des hiérarchies.

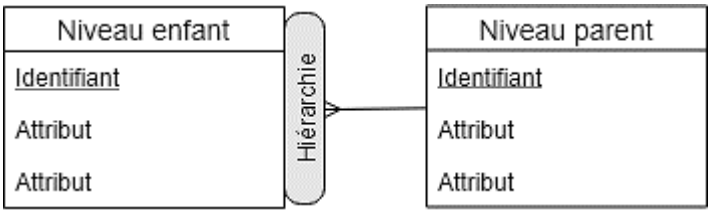


Figure 2.7 : Multidim - Critère d'analyse

Les cardinalités des relations, qui indiquent le nombre minimum et maximum de membres d'un élément (niveau ou fait) pouvant intervenir dans une relation, sont représentées visuellement comme sur la figure 2.8.

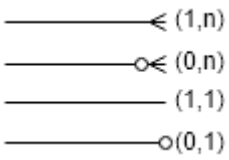


Figure 2.8 : Multidim - Cardinalités des relations

Au sein d'une hiérarchie, le niveau le plus bas, celui qui présente les données sous leur forme la plus détaillée, est appelé le niveau feuille. Le niveau le plus haut, celui qui représente la granularité la moins fine, est quant à lui appelé le niveau racine. Ce niveau racine est généralement représenté schématiquement, par un niveau générique "All" ne présentant qu'un

seul membre. Il n'est toutefois pas obligatoire d'inclure ce niveau dans la représentation schématique.

Une dimension reçoit généralement le nom du niveau feuille de la ou des hiérarchie(s) qui la compose(nt), ou du niveau unique qui la symbolise, en cas d'absence de hiérarchie. Il est toutefois dérogé à cette règle de nommage, dans le cas de figure où une même dimension est amenée à jouer plusieurs rôles dans l'analyse. On parle alors d'une "role-playing dimension". Une telle dimension reçoit un nom différent pour chacun de ses rôles. Cette situation est symbolisée par l'existence de plusieurs relations entre le fait et le niveau feuille (cf. figure 2.9).

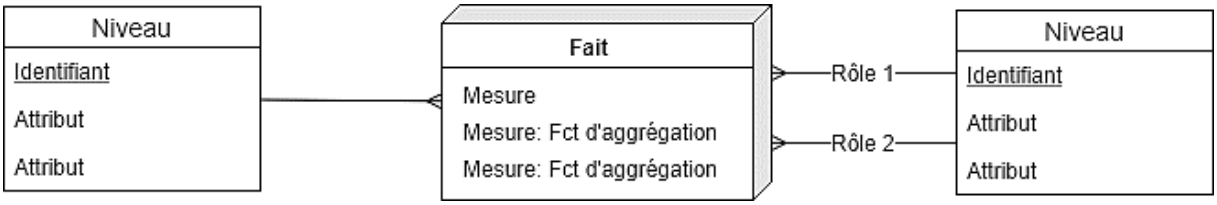


Figure 2.9 : Multidim - Role-playing dimension

Multidim est un modèle présentant une grande expressivité. Il existe en effet une série de symboles, qui rendent possible la représentation de nombreux types de hiérarchies plus ou moins complexes, et qu'il n'est pas possible de représenter dans des schémas logiques. Cela démontre clairement la pertinence de la modélisation conceptuelle. Il est notamment possible d'indiquer que deux relations sont mutuellement exclusives, ce qui permet par exemple, de représenter des hiérarchies dites généralisées. Il s'agit de hiérarchies qui présentent plusieurs chemins d'agrégation (un niveau enfant est en relation avec plusieurs niveaux parents), mais pour lesquelles chaque instance du niveau enfant ne peut s'agréger que suivant un seul de ces chemins (cf. figure 2.10).

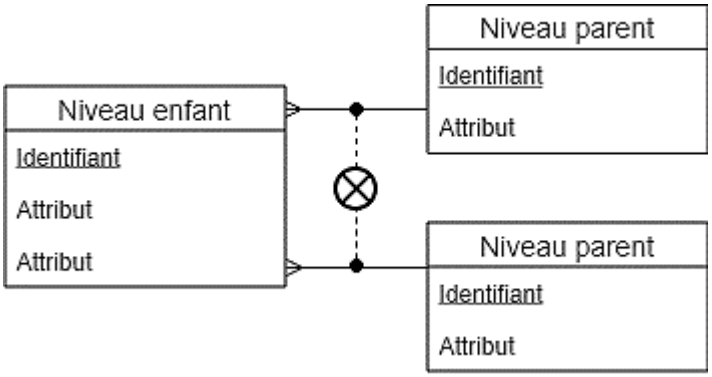


Figure 2.10 : Multidim - Relations mutuellement exclusives

Un type de relations qui mérite une attention particulière en raison des problèmes qu'il peut occasionner en termes d'agrégation, notamment des problèmes de double comptage, est

celui des relations parent-enfant "many-to-many", caractéristique des hiérarchies non-strictes. Ce type de hiérarchies ne permet pas d'observer la propriété de "summarizability" des mesures. En effet, la condition de "disjointness" des instances n'est pas respectée (cf. section 1.4.4).

Il existe différentes façons de résoudre le problème :

- La première façon de faire, consiste à transformer la hiérarchie en hiérarchie stricte. Cela peut notamment se faire en ignorant tous les membres du niveau parent auxquels le membre du niveau enfant est lié à l'exception d'un seul, qui est considéré comme le parent primaire. Une autre façon de procéder, consiste à créer de nouveaux membres au niveau parent, correspondants aux différentes combinaisons des parents individuels associés à un même enfant. De la sorte, chaque instance du niveau enfant se trouve bien liée à un seul membre du niveau parent.

- La deuxième solution nécessite l'introduction d'un facteur de distribution, qui va déterminer la manière dont le membre du niveau enfant et ses mesures doivent être répartis entre les différents parents auxquels ils sont associés (cf. figure 2.11).

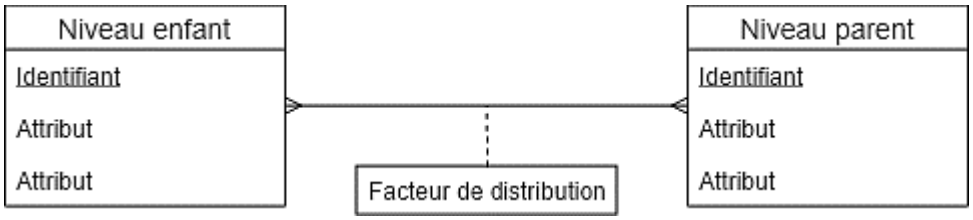


Figure 2.11 : Multidim - Facteur de distribution

- Enfin, une dernière solution consiste à transformer cette hiérarchie en plusieurs dimensions indépendantes. Cette solution a toutefois pour conséquence de changer le cœur de l'analyse. En effet, ajouter une dimension modifie la nature des faits. Notons également que cette solution n'est applicable, qu'en cas de connaissance exacte de la manière dont les mesures doivent être distribuées à la suite du changement de définition des faits.

Remarque : Lorsque des éléments qui pourraient constituer les différents niveaux d'une hiérarchie sont regroupés au sein d'un seul et même niveau sur le schéma conceptuel, cela signifie que cette potentielle hiérarchie ne sera pas utilisée à des fins d'agrégation dans le data warehouse.

2.3.4 Faits

En Multidim, les faits disposent d'un symbole qui leur est propre (cf. figure 2.12), et créent le lien entre plusieurs niveaux auxquels ils sont reliés par des relations. Tout comme pour les niveaux de hiérarchies, les instances d'un fait sont appelées les membres de ce fait. Comme énoncé précédemment, un même niveau peut intervenir plusieurs fois pour un même fait, avec différents rôles représentés par différents liens (cf. figure 2.9). Le niveau de granularité initial d'un fait, est désigné par les niveaux avec lesquels il est en relation.

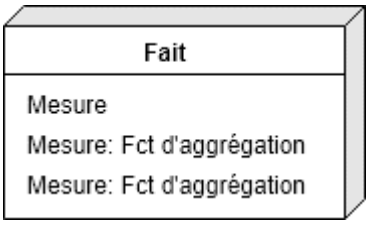


Figure 2.12 : Multidim - Fait

En concordance avec la terminologie du paradigme multidimensionnel, un fait contient des attributs numériques appelés mesures. La fonction d'agrégation associée à une mesure peut être indiquée à côté de celle-ci. Par défaut, les mesures sont supposées additives, et si ce n'est pas le cas, cela doit être indiqué sur le schéma au moyen des symboles adéquats (cf. figure 2.13).

Un cas particulier est celui des mesures dérivées. Il s'agit de mesures pouvant être calculées dynamiquement à partir d'autres mesures ou attributs, et ne devant donc pas être stockées. Elles sont renseignées sur le schéma Multidim au moyen du symbole « / » qui précède leur nom.

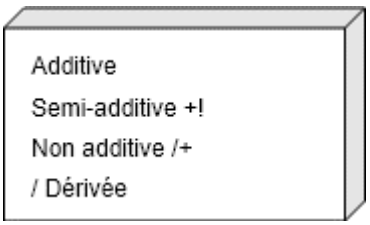


Figure 2.13 : Multidim - Symboles de caractérisation des mesures

2.4 Autres alternatives pour la modélisation conceptuelle

De nombreux autres modèles sont développés dans la littérature, pour la modélisation conceptuelle des data warehouses. Alors que certains auteurs imaginent de nouveaux modèles pensés uniquement pour les données multidimensionnelles, d'autres proposent des extensions

à des langages de modélisation existants et largement utilisés, tels qu'UML, ou encore le modèle entité-association.

Luján-Mora, Trujillo et Song (2006) proposent une extension à UML afin de représenter les propriétés du modèle multidimensionnel au niveau conceptuel. Selon ces auteurs, un premier avantage lié à l'utilisation d'UML vient du fait qu'il s'agisse d'un modèle standard et avec lequel la plupart des designers sont familiers. De ce fait, il n'est pas nécessaire pour ces derniers d'apprendre une nouvelle notation. De plus, UML est une notation extensible. Différents mécanismes permettent d'ajouter de nouveaux éléments afin d'adapter la notation à des domaines particuliers. Enfin, UML permet aussi de tirer avantage de l'expressivité et de la richesse sémantique du paradigme orienté objet.

Franconi et Sattler (1999) ainsi que Tryfona, Busborg et Borch Christiansen (1999) ont pour leur part entrepris d'étendre le modèle entités-associations. Tryfona, Busborg et Borch Christiansen (1999) proposent ainsi la notation starER qui combine la sémantique riche du modèle entités-associations, et la structure en étoile qui s'est imposée dans le domaine des data warehouses. Il est, toujours selon ces auteurs, pertinent de prendre pour point de départ un modèle ayant fait ses preuves, et qui a l'avantage d'être très expressif, simple à utiliser, et facilement extensible. De plus, il n'est pas utile de se détourner d'un modèle aussi largement utilisé et avec lequel la plupart des designers sont familiers.

Un exemple de notation originale, et pensée de bout en bout pour la représentation conceptuelle des systèmes se basant sur le modèle multidimensionnel, au même titre que Multidim, est le Dimensional Fact model proposé par Golfarelli, Maio et Rizzi (1998), et développé en détails par Golfarelli et Rizzi (2009). Cette initiative de création d'une nouvelle notation est partie du postulat qu'il est crucial de procéder à une phase de design conceptuel des systèmes d'information, et du constat que le modèle entités-associations n'est pas adapté à cette phase dans le cas des data warehouses. Selon ces auteurs, les schémas obtenus avec le modèle entités-associations peuvent être compliqués à naviguer et à comprendre pour les utilisateurs. Les schémas de faits, les éléments de base du Dimensional Fact model, permettent une modélisation plus simple, légère, et par conséquent plus facilement abordable par les utilisateurs.

2.5 Design conceptuel du data mart de l'étude de cas avec le modèle Multidim

L'approche suivie pour la phase de design conceptuel de l'étude de cas qui nous intéresse ici, est une approche du type "source-driven". Ce choix est motivé par la mise à

disposition du schéma relationnel de la base de données opérationnelle d'Adventureworks, et par l'impossibilité de procéder à l'élicitation des exigences de futurs utilisateurs, étant donné la nature fictive de l'entreprise. Un dictionnaire des données (Sqldatadictionary.com, 2016) vient également compléter la documentation du système.

2.5.1 Identification des systèmes sources

Dans le cas présent, il est supposé que l'ensemble des données disponibles est fourni par la base de données de l'ERP de l'organisation (cf. figure 1.3 ou Annexe 1). Celle-ci constitue par conséquent, le seul système source du data warehouse.

2.5.2 Application des processus de dérivation

L'application des processus de dérivation repose sur l'analyse du schéma relationnel de la base de données du système opérationnelle, et de sa structure. Cette démarche est effectuée en deux grandes étapes. Dans un premier temps, il convient d'identifier les candidats au rôle de faits, ainsi que leurs mesures candidates associées. Dans un second temps, une fois la recherche des faits et mesures terminée, il est question de découvrir, en analysant la structure et la logique de la source, les dimensions et hiérarchies pouvant être associées à ces faits.

Il est ici pertinent de faire remarquer, que le fait de disposer d'un schéma relationnel accompagné d'une documentation complète et à jour, comme cela est le cas ici, correspond à une situation pouvant être qualifiée d'idéale. Il s'agit toutefois d'un cas de figure assez peu fréquent en pratique. Il est par conséquent généralement souhaitable, dans la mesure du possible, de s'entourer de personnes disposant d'une bonne compréhension des systèmes desquels les données doivent être extraites (Vaisman et Zimányi, 2014). Cela peut permettre de compenser un possible manque de documentation concernant les différentes sources de données, et de grandement faciliter l'évaluation de la qualité des données et l'identification des différents éléments du schéma.

Identification des faits et de leurs mesures associées :

Le processus de recherche de faits et mesures dans le schéma du système opérationnel, nécessite de s'intéresser aux éléments de ce schéma présentant des propriétés dynamiques, c'est-à-dire les éléments qui évoluent rapidement au cours du temps. En d'autres termes, les éléments pouvant potentiellement être retenus en tant que faits pour le data warehouse, sont les éléments de la base de données qui sont les plus fréquemment mis à jour. Il est important d'insister sur la fréquence des changements, car il n'y a en réalité, que très peu d'éléments d'une

base de données qui peuvent être considérés comme véritablement statiques (Golfarelli et Rizzi, 2009).

Les ventes sont le candidat qui se détache clairement dans la situation ici étudiée. Cela est d'autant plus évident, que le data mart développé, l'est à destination du département des ventes d'Adventureworks.

L'enregistrement des données relatives aux ventes réalisées par l'entreprise, est effectué dans les tables Sales.SalesOrderDetail et Sales.SalesOrderHeader, qui sont les deux tables, parmi les tables touchant au domaine des ventes, qui sont le plus fréquemment mises à jour (à chaque enregistrement d'une nouvelle vente). La première de ces deux tables contient les informations spécifiques à chaque vente, alors que la seconde rassemble les informations générales relatives aux commandes, et donc communes à toutes les ventes liées à une même commande.

À ce stade, il peut être mis en avant qu'il existe deux types de ventes distincts pour l'organisation : les ventes réalisées en ligne auprès de particuliers (Internet Sales) et les ventes réalisées par l'intermédiaire des représentants de commerce auprès de revendeurs (Reseller Sales). Ces deux types de ventes sont distingués grâce à l'attribut "OnlineOrderFlag" de la table Sales.SalesOrderHeader, qui prend la valeur "1" lorsque la commande a été passée en ligne, et la valeur "0" dans le cas contraire. Bien qu'en matière de mesures, il n'existe aucune différence entre les deux types de ventes, c'est au niveau des dimensions que des différences se manifestent.

Les mesures qui caractérisent les faits et permettent de les analyser, sont choisies parmi les attributs quantitatifs des deux tables évoquées ci-dessus :

- **OrderQty :** Mesure additive issue de la tables Sales.SalesOrderDetail et renseignant la quantité commandée d'un produit.
- **UnitPrice :** Mesure non-additive agrégée par la moyenne, issue de la table Sales.SalesOrderDetail et renseignant le prix unitaire du produit.
- **UnitPriceDiscount :** Mesure non-additive agrégée par la moyenne, issue de la table Sales.SalesOrderDetail et renseignant la réduction appliquée sur le prix unitaire du produit.

- **Freight :** Mesure additive renseignant les frais de livraison réclamés au client. Il s'agit d'une mesure calculée durant le processus ETL (Extraction, Transformation and Loading), à partir de l'attribut "Freight" de la table Sales.SalesOrderHeader. Les frais de livraison associés à une commande sont répartis entre les différentes ventes de cette commande, proportionnellement à la part qu'elles représentent dans le montant total de la commande.

- **Receipt :** Mesure dérivée et additive renseignant la recette réalisée grâce à une vente. Cette mesure est calculée en soustrayant la réduction unitaire au prix unitaire, et en multipliant ensuite le montant obtenu par la quantité commandée.

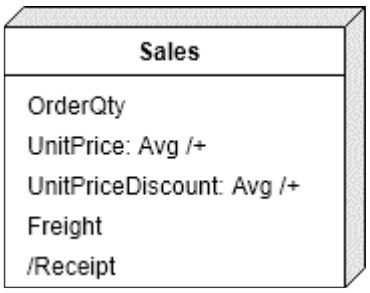


Figure 2.14 : Fait - Sales

Les faits présentent donc les mesures suivantes : la quantité commandée (*OrderQty*), le prix unitaire (*UnitPrice*), la réduction pratiquée sur le prix unitaire (*UnitPriceDiscount*), les frais de livraison (*Freight*), et la recette, qui est une mesure dérivée (/*Receipt*).

Remarque : Il est important de comprendre la différence entre les mesures calculées lors du processus ETL, et les mesures dérivées. Alors que les premières, bien qu'obtenues à la suite de calculs exécutés lors du chargement des données, sont stockées dans le data warehouse au même titre que les mesures disponibles directement, les secondes ne sont pas stockées. Ces dernières sont calculées dynamiquement à partir d'autres mesures et attributs lors de l'exécution des requêtes.

Identification des dimensions :

L'identification des dimensions du schéma, se base sur la recherche d'éléments statiques associés aux faits. Ici, le caractère statique de ces éléments est à appréhender, relativement au caractère hautement dynamique des faits. Beaucoup ne sont en effet pas véritablement statiques au sens strict du terme, et évoluent (sont mis à jour) mais à un rythme généralement bien moindre que les faits.

Une fois qu'un tel élément est identifié, l'étape suivante consiste à l'analyser ainsi qu'à analyser les éléments qui l'entourent dans le système source, et avec lesquels il est en relation. Procéder à cette analyse des liens et attributs du schéma, doit permettre de révéler de potentielles hiérarchies dissimulées dans les données.

Comme expliqué précédemment, c'est au niveau des dimensions que les différences entre les deux types de ventes se manifestent. La différence de nature entre les ventes réalisées en ligne, et les ventes réalisées hors ligne, a pour conséquence l'existence des différences non négligeables en ce qui concerne les informations stockées à leur sujet. Alors que certaines dimensions sont communes aux deux types de ventes, d'autres sont spécifiques à un type de vente en particulier.

Dimensions communes :

La première dimension commune aux deux types de ventes qu'il est possible d'identifier, est la dimension **"Product"**, qui renseigne le produit concerné par la vente. Sur cette dimension se greffe une hiérarchie se déclinant en trois niveaux, la hiérarchie "Categories".

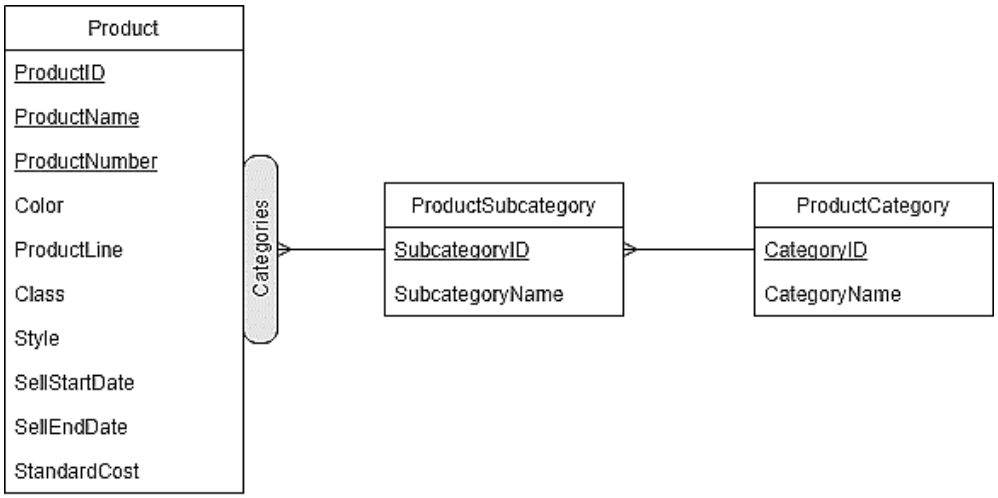


Figure 2.15 : Dimension - Product

Une autre dimension commune à toutes les ventes est la dimension **"SpecialOffer"**, sur laquelle se greffe la hiérarchie "Promotions". Cette dimension contient les informations relatives à l'offre spéciale appliquée à une vente.

En ce qui concerne cette dimension, l'attribut "Category" de la table Sales.SpecialOffer aurait également pu constituer un niveau de la hiérarchie "Promotions". Cependant, celui-ci contient l'information du type de client auquel la promotion est appliquée, ce qui est redondant

avec l'information fournie par l'attribut "OnlineOrderFlag" de la table Sales.SalesOrderHeader qui permet déjà de faire la distinction entre les deux types de ventes.

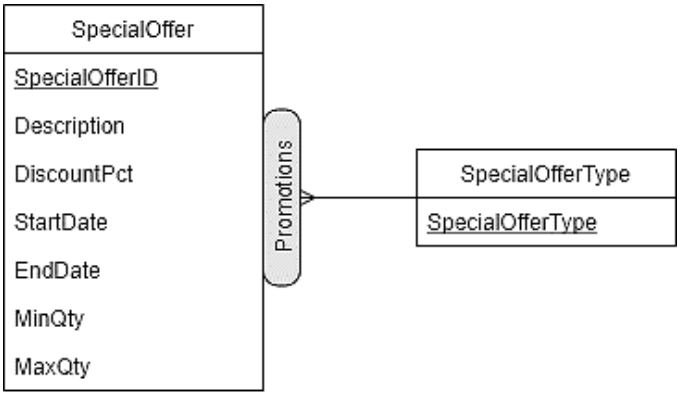


Figure 2.16 : Dimension - SpecialOffer

La dernière dimension commune aux ventes en lignes et aux ventes hors ligne, est une "role playing dimension" qui se décline sous trois noms/rôles : **"OrderDate"** (date à laquelle la commande est passée), **"ShipDate"** (date d'expédition de la commande) et enfin, **"DueDate"** (date limite de paiement pour la commande). Il s'agit d'une dimension temporelle présentant la hiérarchie "Calendar".

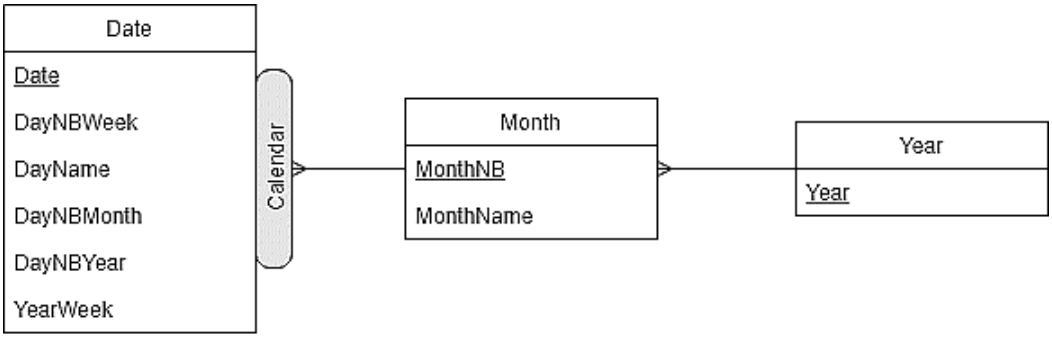


Figure 2.17 : Dimension - OrderDate/ShipDate/DueDate

Dimensions propres aux ventes en ligne :

La dimension **"Order"** n'est pas véritablement une dimension propre aux commandes réalisées en ligne, étant donné que chaque vente appartient à une commande. Cependant, pour les ventes réalisées en ligne, une information supplémentaire est disponible. Il s'agit de la raison d'achat pour le client. Du fait de cette information additionnelle, la dimension "Order" présente, pour les ventes réalisées en ligne uniquement, la hiérarchie optionnelle et non stricte "OrderReason". Cette hiérarchie est à la fois optionnelle, car certaines commandes peuvent n'être reliées à aucune raison d'achat, et non stricte, car une même commande peut être reliée à plusieurs raisons d'achat.

Le problème de la hiérarchie non stricte est résolu grâce à l'utilisation d'un facteur de distribution. La quantité commandée, la recette, et les frais de livraison d'une commande sont répartis de manière égale entre les différentes raisons d'achat associées à cette commande. Par exemple, pour une commande associée à deux raisons d'achat, 50% de la recette est attribuée à chacune de ces dernières.

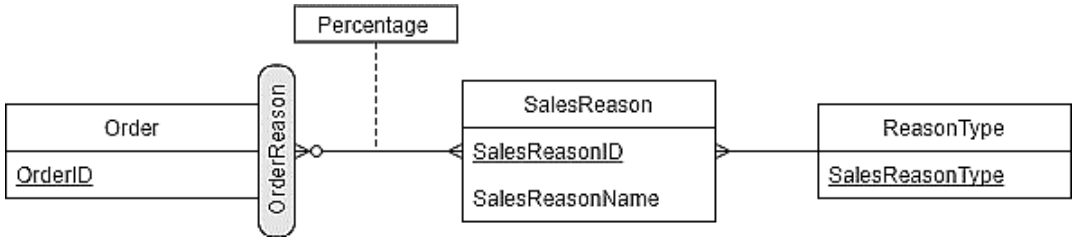


Figure 2.18 : Dimension - Order (Ventes en ligne)

La seule dimension véritablement propre aux ventes en ligne, est la dimension **"Person",** qui rassemble les données relatives à la personne auprès de laquelle la vente a été réalisée. Cette dimension présente une hiérarchie à trois niveaux, la hiérarchie "Territory".

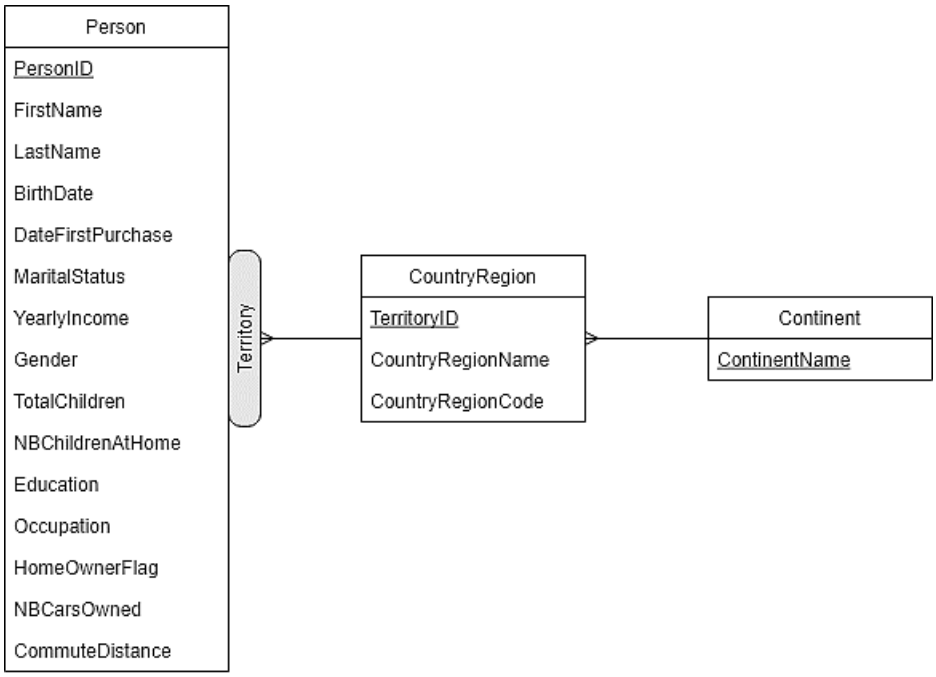


Figure 2.19 : Dimension - Person

Dimensions propres aux ventes hors ligne :

Comme expliqué ci-dessus, la dimension **"Order"** est en réalité commune aux deux types de ventes, mais pour les ventes réalisées auprès de revendeurs, celle-ci est dépourvue de hiérarchie et n'est donc constituée que d'un seul niveau.

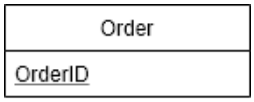


Figure 2.20 : Dimension - Order (Ventes hors ligne)

Une première dimension propre aux ventes hors ligne, est la dimension **"Reseller"** qui rassemble les données relatives au revendeur auprès duquel la vente a été réalisée. Cette dimension partage sa hiérarchie "Territory" avec la dimension "Person", propre à l'autre type de ventes, et présentée précédemment.

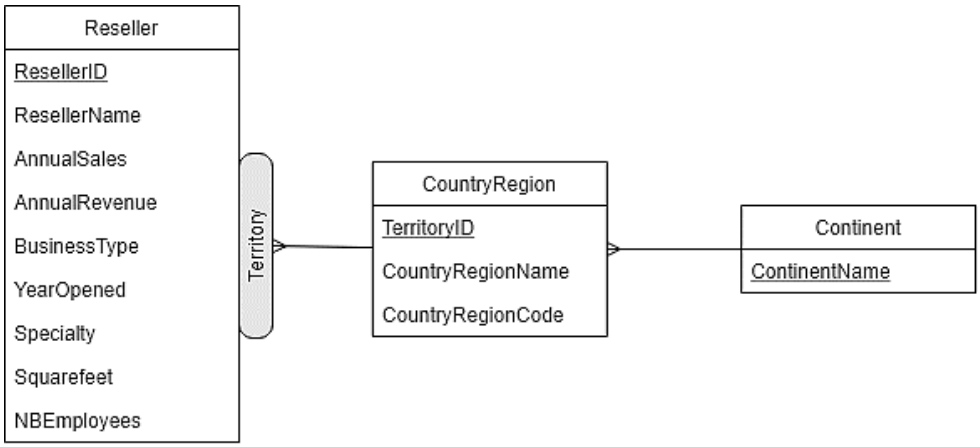


Figure 2.21 : Dimension - Reseller

Enfin, la seconde dimension propre aux ventes hors ligne est la dimension **"SalesPerson"**, qui est dépourvue de toute hiérarchie. Celle-ci fournit les informations concernant le représentant de commerce responsable d'une vente.

Figure 2.22 : Dimension - SalesPerson

En résumé, une vente en ligne est associée à une offre spéciale, un produit, un client, une commande, une date de commande, une date d'expédition, et une date limite de paiement. Une vente hors ligne est pour sa part, associée à une offre spéciale, un produit, un revendeur, une commande, un représentant commercial, une date de commande, une date d'expédition, et une date limite de paiement.

Remarque : Toutes les dimensions du schéma sont des dimensions "one-to-many". En effet, pour chacune d'elle, chaque vente est associée à un seul membre du niveau feuille, alors que chaque membre du niveau feuille peut être lié à plusieurs ventes.

2.5.3 Documentation de la spécification des exigences

Le premier élément inclus dans cette documentation, est un tableau récapitulatif des éléments du modèle multidimensionnel (faits, mesures, dimensions et hiérarchies), identifiés durant la phase de mise en œuvre des processus de dérivation. Ce tableau offre un premier aperçu de la structure du data mart (cf. tableau 2.1).

La deuxième partie de la documentation, est constituée par le schéma conceptuel en lui-même. Concernant ce schéma, étant donné les différences en matière de dimensions entre les deux types de ventes, et pour des raisons de clarté et de lisibilité, la décision a été prise de le scinder en deux parties. Un premier schéma s'intéresse donc aux ventes réalisées en lignes auprès de particuliers (cf. figure 2.23), et un second, aux ventes réalisées hors ligne auprès de revendeurs (cf. figure 2.24).

La troisième et dernière partie de cette documentation, est un mapping entre les éléments du schéma conceptuel du data mart, et les tables du système source (cf. tableau 2.2). Ce mapping met en relation l'origine des données dans les tables de la base de données source, et leur destination dans le data mart. Il précise aussi si une transformation doit être effectuée sur les données avant qu'elles ne soient introduites dans ce dernier.

Bien que les transformations en question soient détaillées lors de la présentation du Processus ETL (cf. chapitre 5), il est ici pertinent, à titre d'exemple, de s'intéresser à l'attribut "BirthDate" du niveau "Person". Les données de cet attribut proviennent du champ "Demographics" de la table Person.Person. Ce champ contient, pour chaque client, un document XML qui renferme une série d'informations démographiques à son sujet, dont sa date de naissance. La transformation consiste simplement en l'extraction de cette dernière du document XML.

Cette dernière pièce trouve sa place dans la documentation de la phase de design conceptuel, car elle permet d'éviter des efforts inutiles lors des phases suivantes. Il s'agit en réalité de métadonnées particulièrement utiles lors du développement du processus ETL.

Vaisman et Zimányi (2014) recommandent également d'inclure à cette documentation, une description complète des systèmes sources. Cela n'a pas été fait ici, mais une telle description est pour rappel, disponible dans le dictionnaire des données de la base de données (Sqldatadictionary.com, 2016).

2.5.4 Validation par les utilisateurs

En théorie, les éléments de la documentation de la spécification des exigences, et en particulier le schéma conceptuel, doivent être soumis à la validation des utilisateurs. Comme expliqué précédemment (cf. section 2.2), cette étape doit permettre aux utilisateurs d'évaluer la pertinence du schéma relativement à leurs besoins d'analyse, et d'identifier d'éventuels éléments manquants ou superflus. Cela n'étant pas possible ici, il est considéré que le schéma conceptuel initial fait également office de schéma conceptuel final.

Tableau 2.1 : Récapitulatif des éléments du modèle multidimensionnel pour le data mart des ventes

Faits	Mesures	Dimensions et cardinalités	Hiérarchies et niveaux
InternetSales	OrderQty UnitPrice (Avg /+) UnitPriceDiscount (Avg /+) Freight /Receipt	Product (1,n)	**Categories** Product → ProductSubcategory → ProductCategory
		SpecialOffer(1,n)	**Promotions** SpecialOffer → SpecialOfferType
		OrderDate (1,n)	**Calendar** Date → Month → Year
		ShipDate (1,n)	**Calendar** Date → Month → Year
		DueDate (1,n)	**Calendar** Date → Month → Year
		Order (1,n)	**OrderReason** Order ↔ SalesReason → ReasonType
		Person (1,n)	**Territory** Person → CountryRegion → Continent
ResellerSales	OrderQty UnitPrice (Avg /+) UnitPriceDiscount (Avg /+) Freight /Receipt	Product (1,n)	**Categories** Product → ProductSubcategory → ProductCategory
		SpecialOffer (1,n)	**Promotions** SpecialOffer → SpecialOfferType
		OrderDate (1,n)	**Calendar** Date → Month → Year
		ShipDate (1,n)	**Calendar** Date → Month → Year
		DueDate (1,n)	**Calendar** Date → Month → Year
		Order (1,n)	/
		Reseller (1,n)	**Territory** Reseller → CountryRegion → Continent
		SalesPerson (1,n)	/

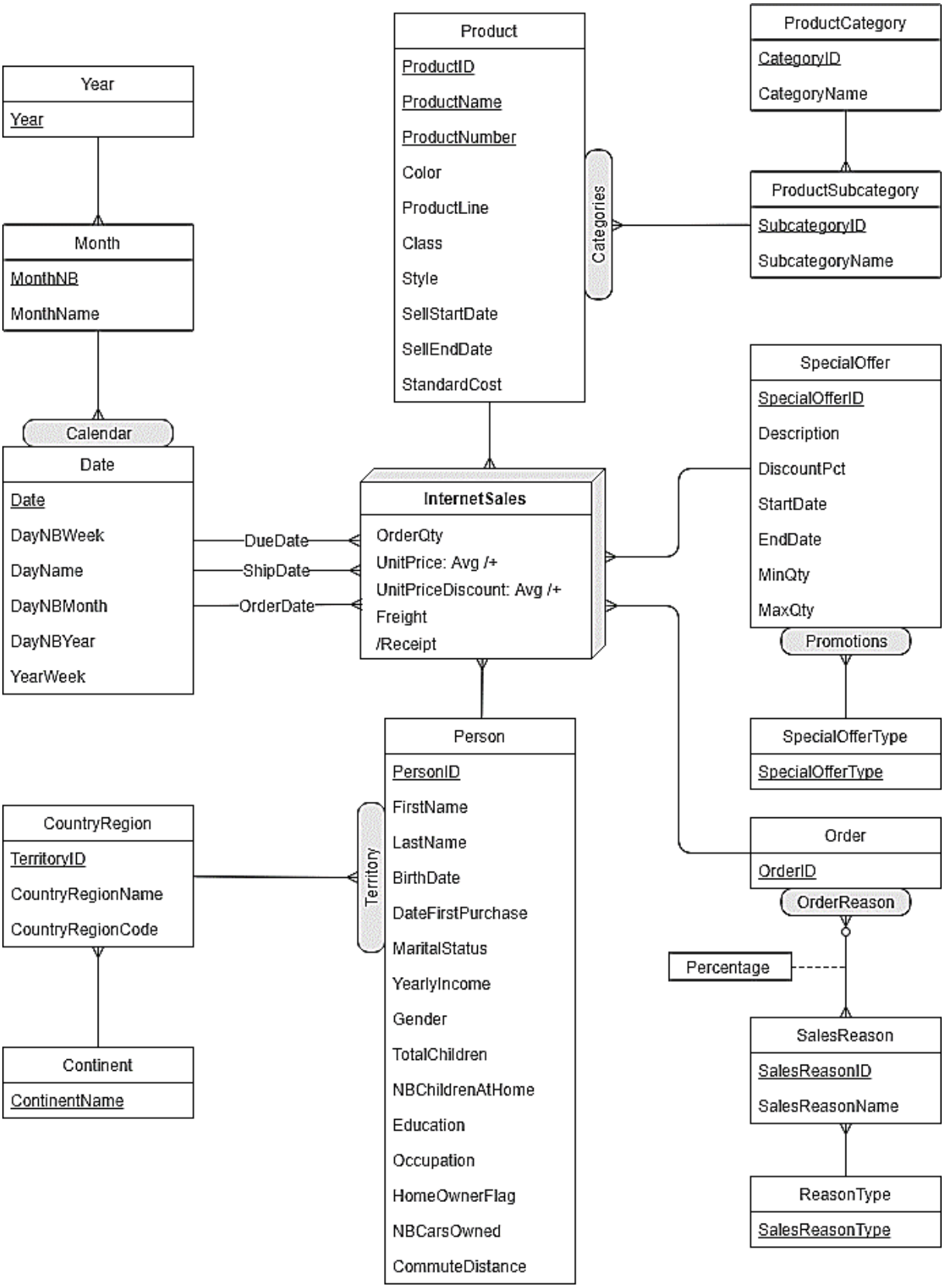


Figure 2.23 : Schéma conceptuel du data mart des ventes (Ventes en ligne)

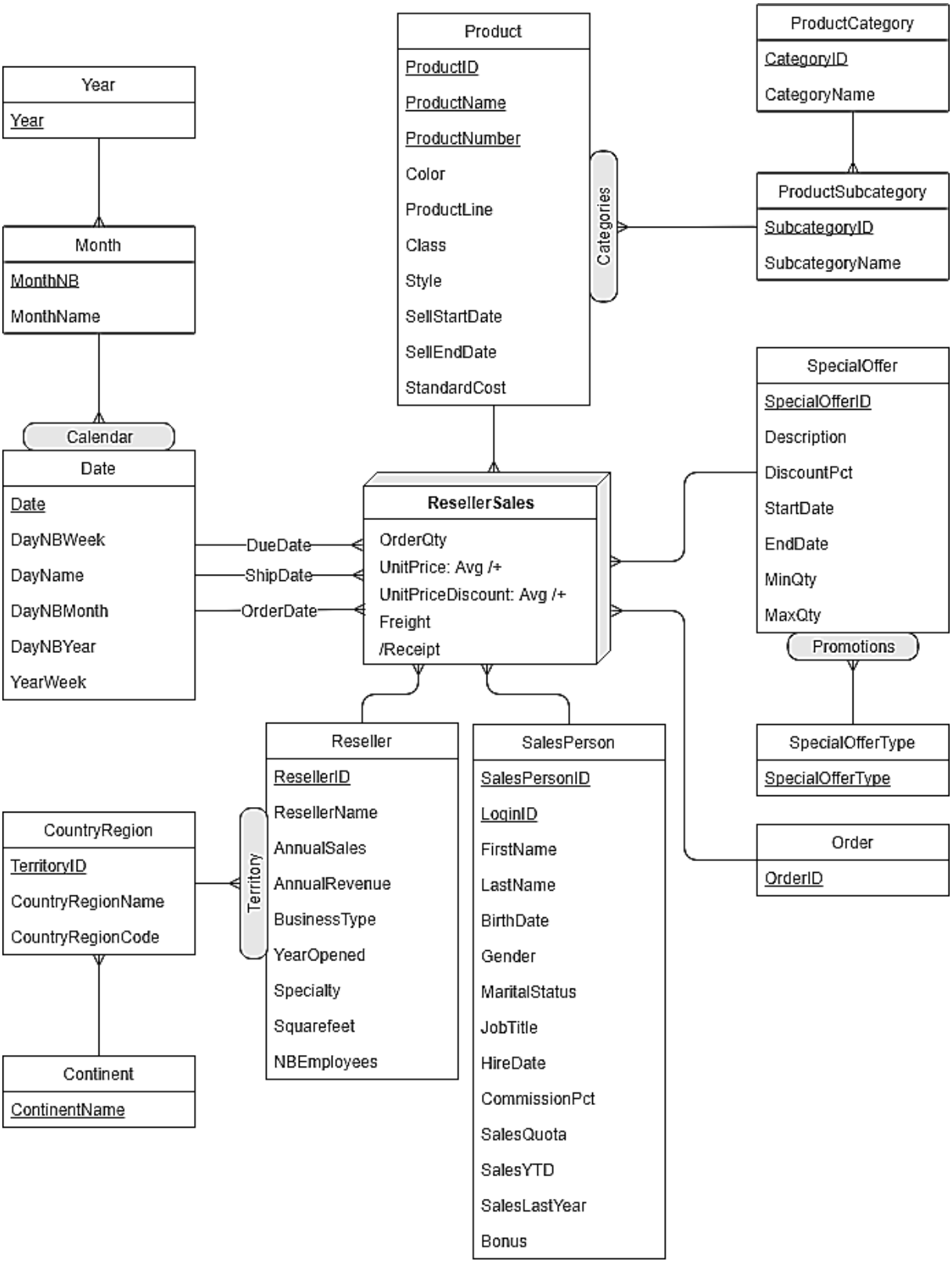


Figure 2.24 : Schéma conceptuel du data mart des ventes (Ventes hors ligne)

Tableau 2.2 : Mapping entre le système source et le data mart

Niveau du Data Warehouse	Attribut du Data Warehouse	Table source	Attribut source	Transformation
Dimension "Product"				
Product	ProductID	Production.Product	ProductID	–
Product	ProductName	Production.Product	Name	–
Product	ProductNumber	Production.Product	ProductNumber	–
Product	Color	Production.Product	Color	–
Product	ProductLine	Production.Product	ProductLine	–
Product	Class	Production.Product	Class	–
Product	Style	Production.Product	Style	–
Product	SellStartDate	Production.Product	SellStartDate	–
Product	SellEndDate	Production.Product	SellEndDate	–
Product	StandardCost	Production.Product	StandardCost	–
ProductSubcategory	SubcategoryID	Production.ProductSubcategory	ProductSubcategoryID	–
ProductSubcategory	SubcategoryName	Production.ProductSubcategory	Name	–
ProductCategory	CategoryID	Production.ProductCategory	ProductCategoryID	–
ProductCategory	CategoryName	Production.ProductCategory	Name	–
Dimension "SpecialOffer"				
SpecialOffer	SpecialOfferID	Sales.SpecialOffer	SpecialOfferID	–
SpecialOffer	Description	Sales.SpecialOffer	Description	–
SpecialOffer	DiscountPct	Sales.SpecialOffer	DiscountPct	–
SpecialOffer	StartDate	Sales.SpecialOffer	StartDate	–
SpecialOffer	EndDate	Sales.SpecialOffer	EndDate	–
SpecialOffer	MinQty	Sales.SpecialOffer	MinQty	–
SpecialOffer	MaxQty	Sales.SpecialOffer	MaxQty	–
SpecialOfferType	SpecialOffertype	Sales.SpecialOffer	Type	–
Dimension "Order"				
Order	OrderID	Sales.SalesOrderHeader	SalesOrderID	–
SalesReason	SalesReasonID	Sales.SalesReason	SalesReasonID	–
SalesReason	SalesReasonName	Sales.SalesReason	Name	–
ReasonType	SalesReasonType	Sales.SalesReason	ReasonType	–
Dimension "Person" et "Reseller"				
Person	PersonID	Person.Person	BusinessEntityID	–
Person	FirstName	Person.Person	FirstName	–
Person	LastName	Person.Person	LastName	–
Person	BirthDate	Person.Person	Demographics	✓
Person	DateFirstPurchase	Person.Person	Demographics	✓
Person	MaritalStatus	Person.Person	Demographics	✓
Person	YearlyIncome	Person.Person	Demographics	✓
Person	Gender	Person.Person	Demographics	✓
Person	TotalChildren	Person.Person	Demographics	✓
Person	NBChildrenAtHome	Person.Person	Demographics	✓
Person	Education	Person.Person	Demographics	✓
Person	Occupation	Person.Person	Demographics	✓
Person	HomeOwnerFlag	Person.Person	Demographics	✓
Person	NBCarsOwned	Person.Person	Demographics	✓
Person	CommuteDistance	Person.Person	Demographics	✓
CountryRegion	TerritoryID	Sales.SalesTerritory	TerritoryID	–

CountryRegion	CountryRegionName	Sales.SalesTerritory	Name	–
CountryRegion	CountryRegionCode	Sales.SalesTerritory	CountryRegionCode	–
Continent	ContinentName	Sales.SalesTerritory	Group	–
Reseller	ResellerID	Sales.Store	BusinessEntityID	–
Reseller	ResellerName	Sales.Store	Name	–
Reseller	AnnualSales	Sales.Store	Demographics	✓
Reseller	AnnualRevenue	Sales.Store	Demographics	✓
Reseller	BusinessType	Sales.Store	Demographics	✓
Reseller	YearOpened	Sales.Store	Demographics	✓
Reseller	Specialty	Sales.Store	Demographics	✓
Reseller	Squarefeet	Sales.Store	Demographics	✓
Reseller	NBEmployees	Sales.Store	Demographics	✓
Dimension "SalesPerson"				
SalesPerson	SalesPersonID	Sales.SalesPerson	BusinessEntityID	–
SalesPerson	LoginID	HumanResources.Employee	LoginID	–
SalesPerson	FirstName	Person.Person	FirstName	–
SalesPerson	LastName	Person.Person	LastName	–
SalesPerson	BirthDate	HumanResources.Employee	BirthDate	–
SalesPerson	Gender	HumanResources.Employee	Gender	–
SalesPerson	MaritalStatus	HumanResources.Employee	MaritalStatus	–
SalesPerson	JobTitle	HumanResources.Employee	JobTitle	–
SalesPerson	HireDate	HumanResources.Employee	HireDate	–
SalesPerson	CommissionPct	Sales.SalesPerson	CommissionPct	–
SalesPerson	SalesQuota	Sales.SalesPerson	SalesQuota	–
SalesPerson	SalesYTD	Sales.SalesPerson	SalesYTD	–
SalesPerson	SalesLastYear	Sales.SalesPerson	SalesLastYear	–
SalesPerson	Bonus	Sales.SalesPerson	Bonus	–
Fait "Sales"				
Sales	OrderQty	Sales.SalesOrderDetail	OrderQty	–
Sales	UnitPrice	Sales.SalesOrderDetail	UnitPrice	–
Sales	UnitPriceDiscount	Sales.SalesOrderDetail	UnitPriceDiscount	–
Sales	Freight	Sales.SalesOrderHeader	Freight	✓
Sales	Receipt	Mesure dérivée		

Remarque : La dimension temporelle n'est pas reprise dans le mapping, car il s'agit d'une dimension générée à partir de rien.

Chapitre 3 : Design Logique

Une fois le schéma conceptuel établi, il doit être traduit en schéma logique, pour pouvoir ensuite être implémenté. C'est à cette étape de design logique que s'intéresse ce chapitre. La première partie du chapitre, consiste en une brève présentation des trois grands types d'architectures, qui peuvent être mis en place dans le cadre du design logique d'un data warehouse. Le chapitre se focalise ensuite sur le modèle ROLAP, avec une présentation des différentes formes que peut prendre une telle architecture, ainsi que des règles de traduction d'un schéma Multidim en schéma ROLAP. Les concepts de "Surrogate key" et de "Slowly Changing Dimensions" sont également abordés. Enfin, le chapitre se termine par la modélisation logique du data mart de l'étude cas.

3.1 Les différentes approches de la modélisation logique d'un data warehouse

Il existe différentes manières d'approcher la modélisation logique d'un data warehouse. En effet, trois modèles, trois grands types d'architectures, peuvent être utilisés afin de représenter des données multidimensionnelles, suivant la manière dont celles-ci vont être stockées : le modèle "Relational OLAP" ou ROLAP, le modèle "Multidimensional OLAP" ou MOLAP et le modèle "Hybrid OLAP" ou HOLAP.

3.1.1 Le modèle Relational OLAP (ROLAP)

Ce premier modèle est celui sur lequel la plupart des solutions commerciales sont basées (Golfarelli et Rizzi, 2009). Avec ce type d'architectures, les données sont stockées dans une base de données relationnelle. Il est donc fait usage du modèle relationnel et de structures de données bidimensionnelles, à savoir des tables, pour le stockage de données multidimensionnelles.

Le principal reproche fait à cette approche, concerne la performance des systèmes. La structure relationnelle, n'est en effet pas la mieux adaptée au stockage des données de nature multidimensionnelle, et à l'exécution de requêtes d'agrégation complexes sur celles-ci. C'est la raison pour laquelle les systèmes ROLAP utilisent des extensions du langage SQL (telles que l'extension SQL OLAP), ainsi que des méthodes d'accès spéciales (pré-calcul de certaines agrégations, index, etc.), afin d'être en mesure d'implémenter efficacement le modèle multidimensionnel (Vaisman et Zimányi, 2014). Toutes ces optimisations permettent généralement d'atteindre un niveau de performance satisfaisant, malgré le stockage sous-

formes de tables, les opérations de jointure qui l'accompagnent, et le haut niveau général de complexité des requêtes SQL devant être exécutées.

Un premier argument en faveur de l'architecture ROLAP, est le fait que le modèle relationnel fasse office de standard dans le monde des bases de données, et qu'il est par conséquent maitrisé par la plupart des designers. De plus, le fait qu'il s'agisse d'un modèle utilisé depuis longtemps, a pour conséquence que les systèmes qui l'utilisent, sont souvent hautement sophistiqués et bien optimisés. Au contraire, les systèmes MOLAP, qui sont apparus plus récemment, n'ont pas encore reçu autant d'attention de la part des scientifiques et de l'industrie. Ils présentent donc un niveau d'évolution bien moindre.

Un autre grand atout des systèmes ROLAP, est le grand espace de stockage qu'ils offrent. Ces systèmes sont en effet beaucoup plus "scalable" que ne le sont les systèmes MOLAP, car ils ne souffrent pas du même problème de "sparsity" (cf. section 3.1.2). Pour reprendre l'image du cube, seules les cellules non vides sont stockées dans les tables. Il n'y a donc pas de gaspillage d'espace pour le stockage de faits ne s'étant pas réellement déroulés ((Vaisman et Zimányi, 2014) ; (Golfarelli et Rizzi, 2009)).

3.1.2 Le modèle Multidimensional OLAP (MOLAP)

Dans le cadre de ce modèle, le "data cube" ne fait pas uniquement office d'abstraction permettant de mieux comprendre les données multidimensionnelles. Il s'agit également du modèle logique du système. Les données sont littéralement stockées dans des structures multidimensionnelles s'apparentant à des cubes, et les opérations OLAP sont directement réalisées sur ces structures.

Il s'agit de la façon la plus simple de représenter et de stocker des données multidimensionnelles. Les opérations OLAP ne doivent pas être traduites en SQL et sont implémentées de manière simple, naturelle, et efficace, directement sur les cubes. La performance de ce type de système est donc bien supérieure à celle des systèmes ROLAP ((Vaisman et Zimányi, 2014) ; (Golfarelli et Rizzi, 2009)).

Ce niveau de performance est toutefois obtenu au détriment de l'espace de stockage. Les structures MOLAP souffrent en effet d'un problème de "sparsity". Contrairement à l'architecture ROLAP, la totalité des cellules du cube (vides et non vides) sont stockées. Étant donné que la proportion de cellules vides peut être très importante, le gaspillage d'espace est non négligeable, et limite la "scalability" de ce type d'architectures. Il existe des solutions permettant de réduire le problème, mais celles-ci provoquent aussi généralement une baisse des

performances du système (Golfarelli et Rizzi, 2009). Toutefois, Colliat (1996) prétend, et a d'ailleurs montré, qu'avec les optimisations adéquates, une architecture MOLAP peut à la fois être plus avantageuse en matière de performances, d'efforts de programmation, et d'espace de stockage.

Si ce manque de capacité de stockage constitue le principal problème des systèmes MOLAP, il n'est pas le seul. Un autre élément qui joue en leur défaveur, est qu'en l'absence d'un standard dominant pour leur implémentation, ils prennent souvent la forme de systèmes propriétaires faiblement documentés. Cela a pour conséquence, d'une part, de les rendre difficilement remplaçables, et d'autre part, de limiter les possibilités en matière d'interopérabilité avec d'autres outils. De plus, même si le langage MDX (Multidimensional Expression language) développé par Microsoft, est de plus en plus largement accepté, le modèle MOLAP ne dispose pas encore d'un langage équivalent à ce que le SQL représente pour le modèle ROLAP. Tous ces éléments tendent encore à générer une certaine méfiance chez les designers, qui préfèrent donc souvent travailler avec le modèle relationnel (Golfarelli et Rizzi, 2009).

3.1.3 Le modèle Hybrid OLAP (HOLAP)

Ce troisième et dernier modèle, constitue, comme son nom l'indique, un compromis entre les deux précédents. Ce type de systèmes tend à bénéficier des avantages des architectures ROLAP et MOLAP, sans souffrir de leurs défauts. L'idée est de bénéficier du grand espace de stockage du premier et de la performance dans le traitement des données du second (Vaisman et Zimányi, 2014). Pour ce faire, une partie des données est stockée dans des tables, alors que l'autre partie est stockée dans des structures multidimensionnelles.

Il existe différentes manières de déterminer quelles données vont être stockées à la manière du modèle ROLAP, et lesquelles vont l'être à la manière du modèle MOLAP (Golfarelli et Rizzi, 2009). Il peut être décidé de stocker les parties les plus denses du cube dans des structures multidimensionnelles, et les parties les moins denses dans des tables. Cette façon de faire apporte un élément de réponse au problème de "sparsity" évoqué précédemment. On peut également choisir de stocker les données détaillées dans des tables, et les agrégations dans des structures MOLAP. Un autre critère pouvant intervenir dans la décision, est la fréquence à laquelle les utilisateurs accèdent aux données. Les données les plus souvent impliquées dans des requêtes, doivent être stockées dans les structures les plus performantes, à savoir les

structures multidimensionnelles, alors que les autres données peuvent être conservées dans des tables.

Selon Vaisman et Zimányi (2014), les systèmes MOLAP ont tendance à être utilisés dans le cas de data marts présentant un faible nombre de dimensions. Au contraire, pour les données à haute dimensionnalité (plus de 10 dimensions), le modèle ROLAP est généralement préféré.

Le modèle ROLAP étant le plus populaire, et celui sur lequel repose la plupart des solutions disponibles sur le marché, c'est à ce modèle que s'intéresse la suite du chapitre. C'est logiquement, également suivant celui-ci qu'est développée la phase de design logique de l'étude de cas.

3.2 Les différentes variantes du modèle ROLAP

Il n'existe pas qu'une seule manière de concevoir une architecture se basant sur le modèle relationnel. Le modèle ROLAP peut en effet prendre la forme de différents schémas. Les deux principales possibilités pour la représentation relationnelle de données multidimensionnelles, sont le Star schema et Snowflake schema. Ces deux modèles ainsi que quelques autres variantes sont présentés ci-dessous.

Notons toutefois que certains auteurs sont opposés à l'utilisation d'un nouveau type de schémas propre aux données multidimensionnelles. Par exemple, Martyn (2004) défend la thèse selon laquelle un data warehouse doit être considéré comme n'importe quelle base de données et, par conséquent, qu'un schéma relationnel classique à la troisième forme normale peut être utilisé, pour l'étape du design logique. Selon lui, bien qu'un tel schéma soit généralement plus complexe, il est aussi plus complet, correcte, stable, et respectueux de la sémantique du domaine d'application. Toujours d'après cet auteur, la volonté d'utiliser un schéma multidimensionnel (Star schema ou Snowflake schema), à des fins de simplification, n'est pas une mauvaise chose en soi. Cependant, une simplification excessive peut cacher une partie importante de la sémantique du domaine, et donc poser des problèmes au niveau de la formulation de requêtes correctes. Il est par conséquent préférable, que le schéma logique reflète la complexité du domaine d'application, et que les utilisateurs s'adaptent à cette complexité. Martyn (2004) recommande donc de formuler dans un premier temps, un schéma relationnel classique, et de s'en éloigner quelque peu par la suite si cela s'avère nécessaire. Il reconnait également, qu'un Snowflake schema peut être perçu comme un compromis entre un Star schema trop simple, et un schéma classique à la troisième forme normale trop complexe.

3.2.1 Le Star schema

Le Star schema est le schéma le plus simple pour la modélisation logique d'un data warehouse. Il est constitué d'une relation centrale appelée la table de faits, et d'un ensemble de relations appelées tables de dimensions (une pour chaque dimension), qui gravitent autour de la table de faits. Ce schéma doit son nom à l'agencement de ses tables, qui rappelle la forme d'une étoile (Datawarehouse4u.info, 2019).

La table de faits contient deux types d'éléments : les mesures, et des clefs étrangères, qui référencent les clefs primaires des tables de dimensions. La clef primaire de la table est la combinaison des clefs étrangères qu'elle contient. Cependant, dans le cas où une autre clef primaire est ajoutée à la table de faits, sous la forme d'une "surrogate key" (cf. section 3.4), cette combinaison devient une clef alternative. La table de faits est une table normalisée. Alors que la clef primaire détermine fonctionnellement les valeurs des mesures, aucune dépendance de la sorte n'existe entre les différentes clefs étrangères (Vaisman et Zimányi, 2014).

Les tables de dimensions contiennent pour leur part, une clef primaire ainsi qu'un ensemble d'attributs décrivant les dimensions. Contrairement à la table de faits, les tables de dimensions ne sont généralement pas normalisées et peuvent, par conséquent, présenter de la redondance. Cette redondance s'explique par le fait que, même en présence d'une hiérarchie, chaque dimension ne soit représentée que par une seule table, alors qu'il existe des dépendances fonctionnelles entre les attributs des différents niveaux d'une hiérarchie. Ces tables peuvent donc contenir des dépendances fonctionnelles autres que celle qui uni leur clef primaire aux autres attributs.

Le premier point positif des Star schémas, est qu'il s'agit d'un type de schémas très simple et facile à comprendre (Datawarehouse4u.info, 2019). Un autre grand atout, est qu'ils facilitent la formulation et l'exécution des requêtes. Le fait que chaque dimension ne comporte qu'une seule table, limite en effet grandement le nombre de jointures à réaliser, et augmente par conséquent la performance du système.

Cette façon de procéder ne présente cependant pas que des avantages. En plus de ne pas permettre la représentation explicite des hiérarchies (ce qui peut les rendre difficiles à appréhender), elle provoque généralement, comme expliqué précédemment, l'existence dans les tables de dimension d'une certaine part de redondance.

On parle cependant de redondance contrôlée. Bien que la non-normalisation des tables de dimension complexifie le processus de chargement des données, et ne soit pas optimal en

matière d'espace de stockage, on considère généralement que les gains de performance justifient ces quelques désagréments. De plus, le gaspillage d'espace dû à la redondance, est d'autant moins préoccupant, que les tables de dimensions occupent, comparativement à la table de faits, un faible espace dans le data warehouse ((Vaisman et Zimányi, 2014) ; (Golfarelli et Rizzi, 2009)).

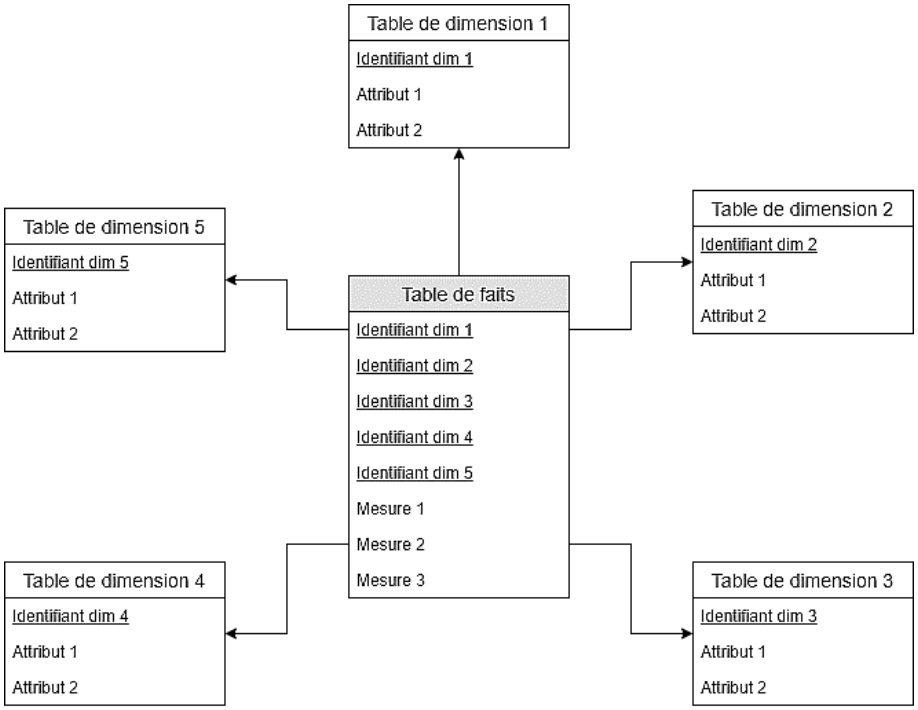


Figure 3.1 : Star schema

3.2.2 Le Snowflake schema

Le Snowflake schema est un schéma très similaire au Star schema, mais qui évite la redondance en normalisant les tables de dimensions. Il peut donc être perçu comme une normalisation du Star schema.

Les dimensions qui comportent une hiérarchie, sont ici représentées par plusieurs tables liées les unes aux autres par des contraintes d'intégrité référentielle. Chaque niveau d'une hiérarchie dispose de sa propre relation, qui comprend une clef primaire, les attributs correspondants à ce niveau et qui dépendent fonctionnellement de la clef primaire, ainsi qu'une clef étrangère qui référence le niveau parent dans la hiérarchie, si un tel niveau existe. Pour sa part, la table de faits contient toujours les mesures, ainsi qu'une série de clefs étrangères qui référencent les niveaux les plus bas des différentes dimensions.

Les tables de dimensions qui correspondent aux niveaux les plus bas des hiérarchies, peuvent être appelées, tables de dimensions primaires. Celles correspondant aux autres niveaux, peuvent être appelées, tables de dimensions secondaires (Golfarelli et Rizzi, 2009).

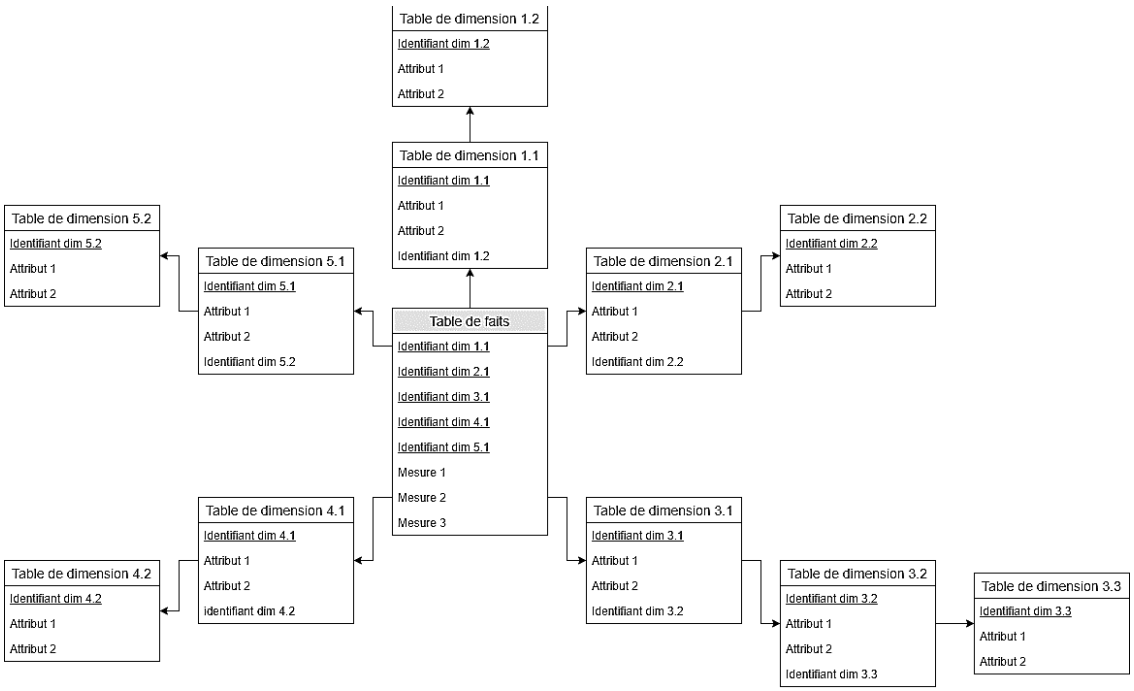


Figure 3.2 : Snowflake schema

La normalisation des tables présente plusieurs avantages. D'abord, l'absence de redondance et les contraintes d'intégrité référentielles, rendent les tables normalisées plus faciles à maintenir. Ensuite, cette facilité de maintenance, s'accompagne également d'une meilleure optimisation de l'utilisation de l'espace de stockage. Un autre point positif est le fait que, ce type de modèles représente de manière explicite les structures hiérarchiques dans les différentes dimensions. Elles peuvent donc plus facilement être comprises. Cela a également pour conséquence, puisque chaque niveau correspond à une relation différente, de permettre à plusieurs hiérarchies de partager un même niveau ((Vaisman et Zimányi, 2014) ; (Golfarelli et Rizzi, 2009)).

Le principal handicap qui accompagne l'utilisation d'une telle architecture, est la perte de performance due à l'augmentation du nombre de jointures devant être réalisées, afin de parcourir les hiérarchies (Vaisman et Zimányi, 2014). Cette perte de performance ne se manifeste toutefois pas, lorsque seules les tables de dimensions primaires sont impliquées dans une requête. Dans ce cas de figure précis, la réduction de la taille des tables de dimensions a même tendance à améliorer les performances du système (Golfarelli et Rizzi, 2009).

3.2.3 Autres variantes

Vaisman et Zimányi (2014) présentent également deux autres variantes de schémas pour le modèle ROLAP : le Starflake schema et le Constellation schema.

Le premier peut être vu comme un schéma hybride, dans lequel certaines dimensions sont normalisées alors que d'autres qui pourraient l'être, ne le sont pas.

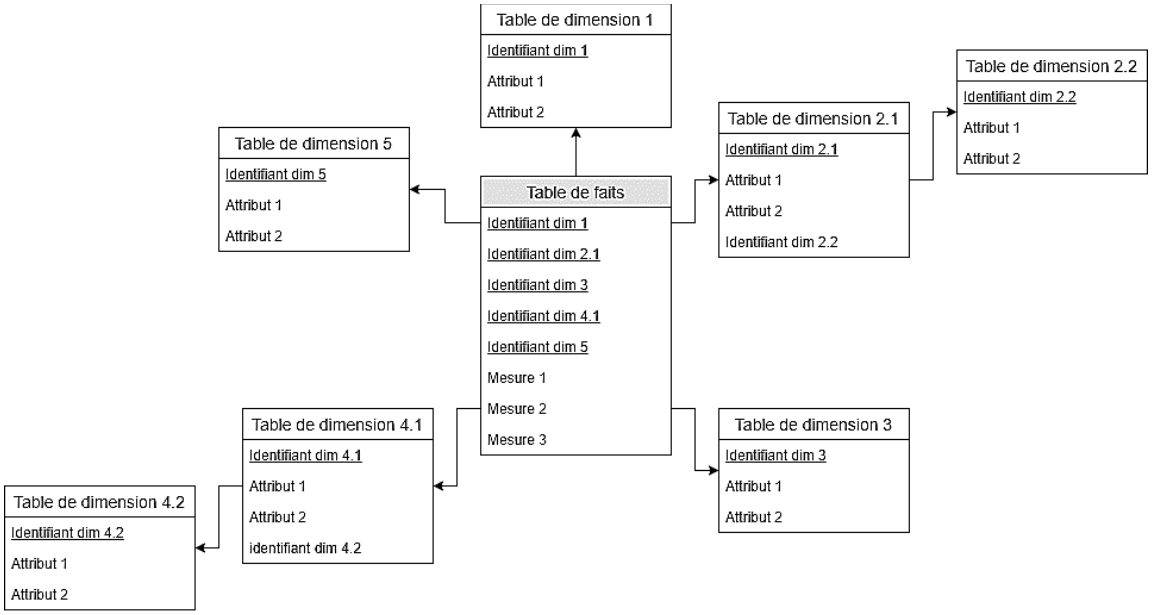


Figure 3.3 : Starflake schema

Le second est un schéma qui comporte plusieurs tables de faits qui partagent certaines dimensions (des dimensions conformes).

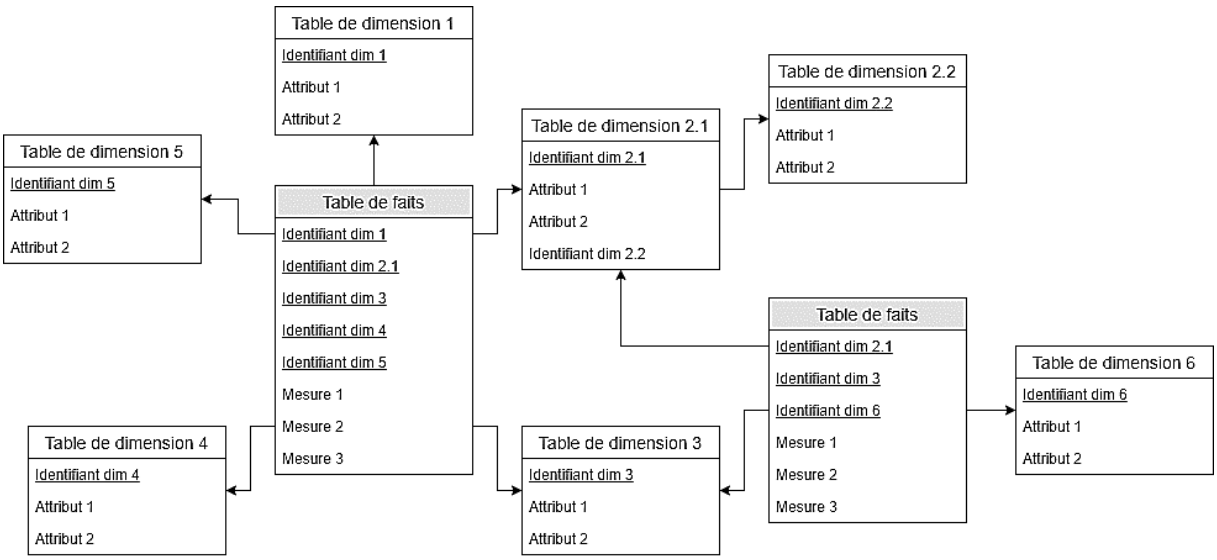


Figure 3.4 : Constellation schema

3.3 Traduction du modèle Multidim vers le modèle ROLAP (Vaisman et Zimányi, 2014)

Afin de traduire un schéma Multidim en schéma relationnel, les créateurs de la notation énoncent une série de règles de traduction. L'application stricte de ces règles, aboutit à la création d'un Snowflake schema. Pour générer un Star schema, il faut rassembler les différentes tables constitutives d'une dimension, en une seule.

3.3.1 Règle 1

Un niveau, à condition qu'il ne soit pas lié à un fait par une relation one-to-one, est transformé en une table qui contient tous les attributs du niveau. Une "surrogate key" peut être ajoutée à la table. Si ce n'est pas le cas, l'identifiant du niveau devient la clef primaire de la table. Des attributs supplémentaires peuvent être ajoutés à la table par application de la règle 3, qui concerne la traduction vers le modèle relationnel, des relations entre les éléments d'un schéma Multidim.

3.3.2 Règle 2

Un fait est transformé en une table, qui contient tous les attributs et toutes les mesures de ce fait. Une "surrogate key" peut être ajoutée à la table. Tout comme pour la règle 1, des attributs supplémentaires peuvent être ajoutés à la table par application de la règle 3, qui concerne la traduction vers le modèle relationnel, des relations entre les éléments d'un schéma Multidim.

3.3.3 Règle 3

Une relation, que celle-ci relie un fait à un niveau d'une dimension, ou le niveau enfant d'une dimension à son niveau parent, peut être traduite de trois façons, en fonction de sa cardinalité :

- Si la relation présente une cardinalité one-to-one, la table correspondant au fait ou au niveau enfant, est étendue avec l'ensemble des attributs du niveau (dans le cas du fait), ou du niveau parent (dans le cas du niveau enfant).

- Si la relation présente une cardinalité one-to-many, la table correspondant au fait ou au niveau enfant, est étendue avec la clef primaire ("surrogate key" ou identifiant) du niveau (dans le cas du fait), ou du niveau parent (dans le cas du niveau enfant). La table de faits ou le niveau comporte ainsi une clef étrangère qui référence respectivement, le niveau ou le niveau parent.

- Si la relation présente une cardinalité many-to-many, une nouvelle table appelée "bridge table" est créée, et contient comme attributs les clefs primaires des tables correspondant au fait et au niveau, ou des tables correspondant au niveau enfant et au niveau parent. La combinaison des deux attributs, constitue la clef primaire de la nouvelle table, et chacun d'eux joue le rôle d'une clef étrangère, qui référence sa table d'origine. Si la relation dispose d'un facteur de distribution, un autre attribut est ajouté à la table afin de stocker cette information.

Remarque : Une dimension liée à un fait par une relation one-to-one dans le schéma Multidim, et qui par conséquent, voit ses attributs incorporés à la table de faits dans le schéma logique, est appelée une "fact dimension" ou une "degenerate dimension".

3.3.4 Hiérarchies non strictes

L'application des règles présentées ci-dessus, au cas d'une hiérarchie non stricte, mène à la création d'une table pour chacun des deux niveaux impliqués, ainsi que d'une troisième table qui matérialise la relation many-to-many entre les deux niveaux, la "bridge table".

Comme expliqué au chapitre précédent (cf. section 2.3.3), il est également possible de se débarrasser d'une hiérarchie non stricte en ajoutant une dimension supplémentaire au schéma. Cette façon de faire permet d'éviter la création d'une "bridge table", mais implique la création d'une ou plusieurs nouvelles tables de dimensions.

En matière d'espace de stockage, la création d'une "bridge table" augmente moins l'espace total occupé par le data warehouse que la création d'une nouvelle dimension. Non seulement la ou les tables de dimensions doivent être stockées, mais l'augmentation du nombre de dimensions s'accompagne d'une augmentation du nombre de clefs étrangères à inclure dans la table de faits, ce qui augmente aussi l'espace nécessaire au stockage de cette dernière. D'un autre côté, le recours à une "bridge table" tend à complexifier l'agrégation des mesures le long d'une hiérarchie (efforts de programmation supplémentaires, augmentation du nombre de jointures, etc.) (Vaisman et Zimányi, 2014).

Vaisman et Zimányi (2014) considèrent que le recours à des "bridge tables" est adéquat, dans les cas de figure où le data warehouse comporte peu de hiérarchies non strictes, et où les facteurs de distribution ne varient pas avec le temps. Dans le cas contraire, la transformation des hiérarchies non strictes en hiérarchies strictes est préférable.

3.4 L'utilisation de Surrogate keys

Dans les paragraphes précédents, la possibilité d'ajouter une "surrogate key" aux différentes tables a plusieurs fois été évoquée. Le principal avantage d'une telle opération pour les tables de dimensions, est la création d'une certaine forme d'indépendance entre le data warehouse et les systèmes sources. Ne pas utiliser les identifiants de ces systèmes comme clefs primaires, permet de protéger le data warehouse d'éventuels changements au niveau de ces identifiants (Vaisman et Zimányi, 2014), et de faciliter la mise en place de "slowly changing dimensions" (cf. section 3.5) (Golfarelli et Rizzi, 2009).

Aussi bien pour les tables de dimensions que pour les tables de faits, l'utilisation de ce type de clefs, puisqu'elles prennent souvent la forme d'entiers, permet aussi d'améliorer les performances du système et de ses index ((Vaisman et Zimányi, 2014) ; (Golfarelli et Rizzi, 2009)).

Malgré l'utilisation de "surrogate keys", il est conseillé de conserver les identifiants issus des systèmes sous-jacents, afin de pouvoir toujours faire correspondre les données dans les deux environnements. Cette pratique augmente cependant la taille des tables, ce qui constitue le premier inconvénient de l'utilisation de telles clefs. Un second inconvénient est l'ajout d'une étape au processus ETL ((Vaisman et Zimányi, 2014) ; (Golfarelli et Rizzi, 2009)).

3.5 Slowly Changing Dimensions (SCD)

Le concept de "Slowly Changing Dimensions" concrétise l'abandon de l'idée que les dimensions d'un data warehouse sont des éléments statiques, qui une fois définis, n'évoluent jamais. Au niveau de la structure des tables, il est notamment possible qu'un attribut soit supprimé ou ajouté dans l'un des systèmes sources, et doive par conséquent également l'être au niveau du data warehouse. Au niveau des instances, il peut arriver qu'une erreur introduite dans une table de dimension doive être corrigée (ex : un produit de la catégorie 1 a faussement été encodé comme appartenant à la catégorie 2). Il existe aussi la possibilité qu'un changement du contexte organisationnel ou d'un scénario d'analyse, nécessite une mise à jour d'une ou plusieurs tables de dimensions (ex : un produit appartenant anciennement à la catégorie 1 appartient maintenant à la catégorie 2). Les dimensions pensées afin de supporter ce type de modifications, sont appelées des "Slowly Changing Dimensions".

En ce qui concerne les modifications au niveau des instances, trois solutions « classiques » sont généralement présentées dans la littérature (Vaisman et Zimányi, 2014) :

- La première solution, ou SCD de type 1, consiste simplement à écraser l'ancienne valeur avec la nouvelle. Il s'agit du type le plus simple de SCD, car aucune trace de la modification ou de l'ancienne valeur n'est conservée. Celle solution convient parfaitement aux situations dans lesquelles une erreur doit être corrigée.

- La deuxième solution, ou SCD de type 2, consiste à ajouter à la table une nouvelle version du tuple concerné par le changement, tout en conservant l'ancienne. L'historique des modifications est donc conservé. C'est ici que l'ajout d'une "surrogate key" prend tout son sens relativement au concept de SCD. En effet, l'utilisation d'une clef indépendante de tout système source permet de facilement conserver différentes versions d'une même ligne, car chacune d'elle peut ainsi avoir sa propre valeur de clef primaire. Avec cette méthode, qui nécessite également l'ajout de deux colonnes contenant les limites de l'intervalle de validité des tuples, les faits contribuent aux agrégations selon le tuple qui était en vigueur au moment de leur enregistrement (ex : si un produit passe de la catégorie 1 à la catégorie 2, les ventes de ce produit antérieures au changement sont agrégées pour la catégorie 1, et celles qui lui sont postérieures sont agrégées pour la catégorie 2).

- La troisième solution, ou SCD de type 3, prévoit pour chaque attribut sujet à une modification, l'ajout d'une colonne contenant la nouvelle valeur de cet attribut. À moins d'ajouter davantage de colonnes, seule la valeur actuelle et la valeur précédente d'un attribut sont conservées. L'historique des modifications n'est donc que partiel.

Il est important de noter qu'il est possible de combiner les différentes solutions présentées ci-dessus. De nombreux auteurs proposent d'ailleurs leurs propres types de SCD, qui correspondent généralement à des variantes et des combinaisons de ces trois types de base.

Remarque : Dans le cas d'une SCD de type 2 au sein d'une architecture "Snowflake", les changements apportés à un niveau d'une hiérarchie doivent être propagés aux niveaux inférieurs.

3.6 Design logique du data mart de l'étude de cas

Pour la modélisation logique du data mart du département des ventes de la société Adventureworks, la décision a été prise d'opter pour un "Starflake schema", dans lequel certaines dimensions sont totalement normalisées, certaines seulement partiellement, et d'autres pas du tout. Le choix de cette architecture hybride est motivé par une volonté de trouver un

compromis, entre l'optimisation du stockage des données d'une part, et la performance du système d'autre part. Il permet également de trouver un équilibre en matière de complexité du processus ETL.

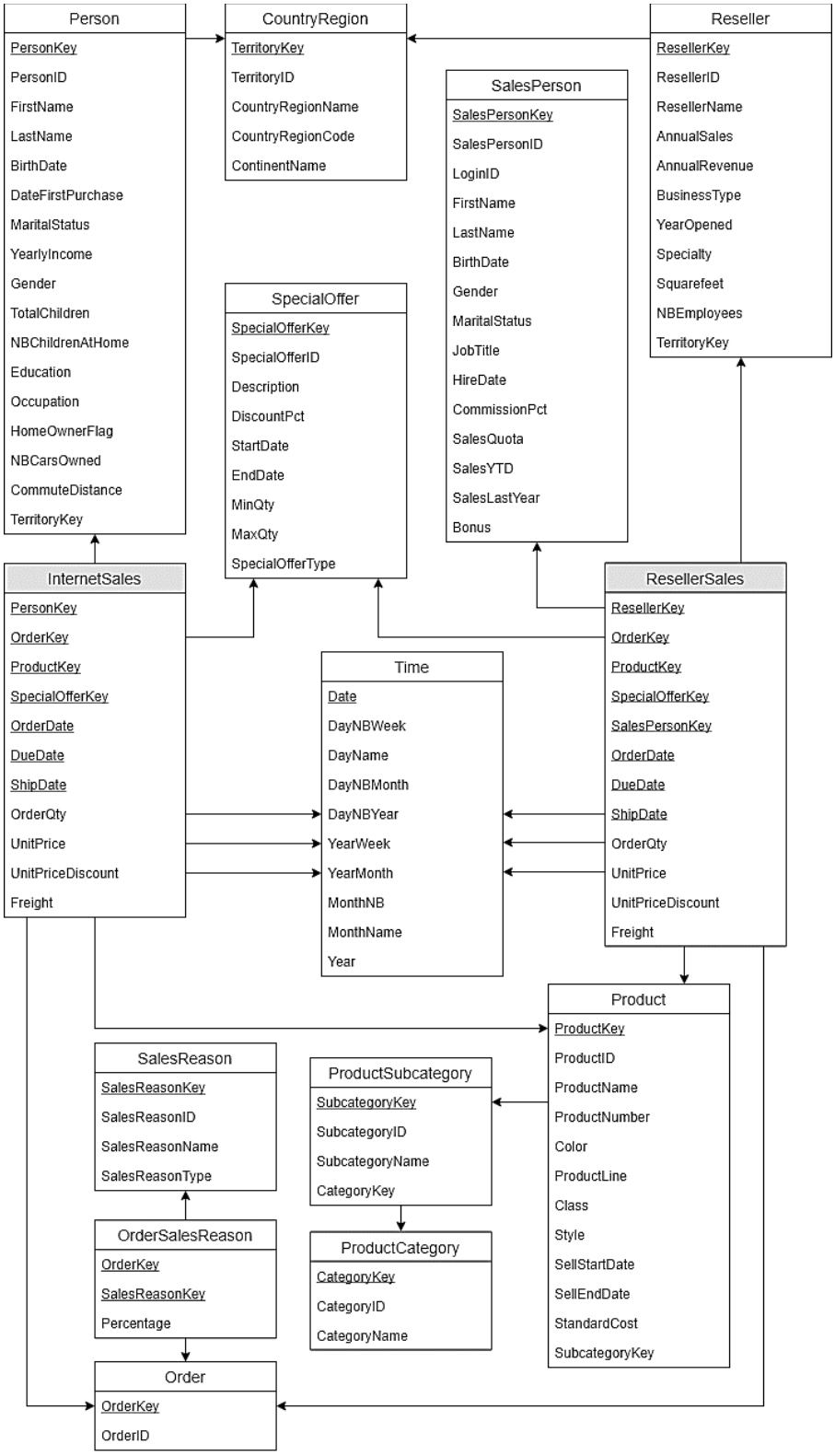


Figure 3.5 : Schéma logique du data mart des ventes

Le schéma obtenu au terme de la phase de design logique est fourni à la figure 3.5. Dans ce schéma, la dimension "Product" est entièrement normalisée. C'est également le cas de la dimension "SalesPerson" puisque celle-ci ne présente pas de structure hiérarchique. Les dimensions "Person", "Reseller" et "Order" ne sont que partiellement normalisées alors que, les dimensions "Time" et "SpecialOffer" ne le sont pas du tout. Il est également important de préciser le fait que toutes les tables de dimensions, à l'exception de la table "Time", se voient ajouter une "surrogate key", mais que les identifiants issus du système source sont conservés.

Le choix de normaliser certaines dimensions, permet l'optimisation du stockage des données sur plusieurs plans. D'une part, la redondance est supprimée des tables des dimensions concernées. D'autre part, cette architecture permet le partage de niveaux entre les dimensions qui ont été normalisées. Ainsi, les données du niveau "CountryRegion", qui intervient à la fois dans la dimension "Person" et dans la dimension "Reseller", ne sont stockées qu'une seule fois et ne doivent pas être dupliquées.

Un élément frappant concernant ce schéma, est le fait que chacun des deux types de ventes (ventes en ligne et ventes hors ligne) dispose de sa propre table de fait, faisant ainsi du schéma un "Constellation schema" (cf. section 3.2.3). Pour rappel, sur le schéma conceptuel, les deux types de ventes avaient été séparés avant tout, pour des raisons de clarté et de lisibilité. Au niveau logique, c'est la différence fondamentale de nature entre les deux types de ventes qui motive leur séparation. Une vente réalisée en ligne auprès d'un particulier peut difficilement être assimilée, sur le plan analytique, à une vente réalisée hors ligne auprès d'un revendeur. Le fait que certaines dimensions soient propres à un type de ventes en particulier, fait en sorte que le contexte et les scénarios d'analyse différent fortement. De plus, conserver tous les faits dans une même table, obligerait à autoriser certains attributs constitutifs de la clef primaire à prendre une valeur nulle (ex : pour toutes les ventes en ligne, la clef étrangère qui référence la table "SalesPerson" prendrait une valeur nulle). Or, cela n'est pas toléré par beaucoup de systèmes de gestion de bases de données. C'est notamment le cas de Microsoft SQL Server, qui est la plateforme d'implémentation du data mart (cf. figure 3.6). Le recours à deux tables de faits permet donc d'éviter une telle situation.

Concernant les tables de faits, il peut encore être précisé qu'elles ne contiennent pas la mesure "Receipt", car celle-ci est une mesure dérivée, et ne nécessite donc pas d'être stockée dans le data mart.

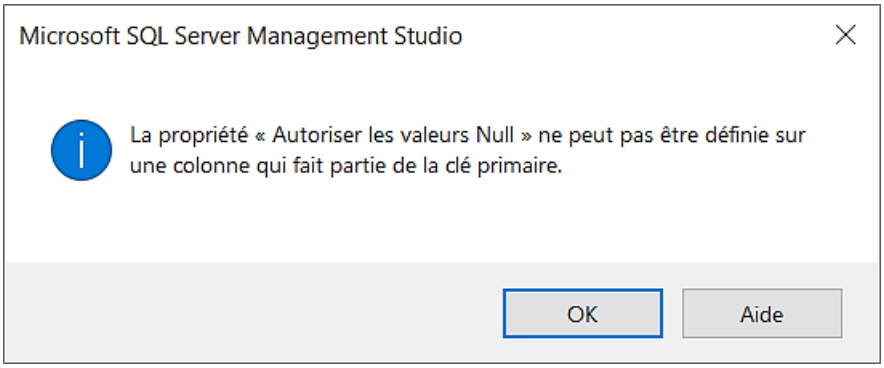


Figure 3.6 : Message d'erreur obtenu avec SQL Server Management Studio lorsqu'un attribut d'une clef primaire est défini comme pouvant être null

Remarque : "Time" est le nom générique qui est donné à la dimension temporelle. Pour rappel, il s'agit d'une "role-playing dimension" à trois rôles : "OrderDate", "DueDate" et "ShipDate".

Chapitre 4 : Design Physique

Dans ce chapitre, la question du design logique d'un data warehouse est abordée sur le plan théorique. Le design physique d'un data warehouse est une étape cruciale du processus d'implémentation, qui consiste à mettre en place une série de méthodes et de mécanismes afin de permettre au système, d'atteindre un certain niveau de performance. Il existe trois grands moyens d'améliorer les performances d'un système OLAP : les vues matérialisées, les index, et le partitionnement (Vaisman et Zimányi, 2014). Ces trois pratiques sont abordées à tour de rôle au cours de ce chapitre.

4.1 Les vues matérialisées

"Une vue n'est rien de plus qu'une requête stockée dans une base de données, à laquelle un nom est associé et qui peut être réutilisée comme n'importe quelle table, mais qui doit être recalculée à chaque fois. Dans le cas d'une vue matérialisée, les données obtenues grâce à la requête sont stockées physiquement dans la base de données." (Vaisman et Zimányi, 2014).

Les vues matérialisées contribuent à l'optimisation des performances d'un système, car elles permettent le précalcule et le stockage du résultat d'opérations complexes, qui ne doivent donc pas être exécutées à chaque fois qu'une requête est lancée. Cette amélioration des performances se paie toutefois logiquement au niveau de l'espace de stockage, puisque davantage de données sont stockées.

Deux grands problèmes se posent dans le cadre de la mise en place de vues matérialisées : celui de la sélection de ces vues, et celui de leur maintenance. Ces deux questions sont d'ailleurs liées, puisque l'effort de maintenance est un facteur qui entre en ligne de compte dans le processus de sélection des vues.

Puisque les vues matérialisées sont construites sur bases de requêtes exécutées sur les tables de base du data warehouse, tout changement apporté à ces tables risque d'avoir un impact sur le contenu des vues, qui doivent donc être mises à jour. C'est la propagation de ces changements réalisés sur les tables de base du data warehouse vers les vues matérialisées, qui constitue la tâche de maintenance de ces dernières (Vaisman et Zimányi, 2014).

L'idéal pour une vue matérialisée, est d'être "self-maintainable". C'est le cas, lorsqu'elle peut être maintenue sans qu'il ne soit nécessaire d'accéder aux données de base du data

warehouse. Une telle vue peut donc être mise à jour en faisant usage uniquement de la vue elle-même, et des contraintes clefs de sa définition.

De manière générale, il existe deux façons de mettre à jour une vue matérialisée. La première méthode consiste à la recalculer totalement à partir de rien. L'autre alternative est la maintenance incrémentale, qui permet d'éviter la lourde tâche de générer à nouveau la totalité des données de la vue (Vaisman et Zimányi, 2014).

En ce qui concerne la question de la sélection des vues à matérialiser, une première approche consiste à matérialiser la totalité du cube, autrement dit, à créer des vues matérialisées pour toutes les agrégations possibles. Cette manière de procéder garantit que toute les requêtes d'agrégation correspondent forcément aux données d'une des vues. Le temps de réponse est par conséquent grandement réduit. Cependant, la création d'un aussi grand nombre de vues matérialisées pose des problèmes, tant au niveau de leur stockage que de leur maintenance.

La seconde approche est aux antipodes de la première. Elle consiste à ne créer aucune vue matérialisée. C'est logiquement la manque de performance du système, qui rend cette approche généralement peu viable.

La troisième approche constitue un compromis entre les deux premières. Seule une partie des vues est effectivement matérialisée. C'est dans ce contexte, que se pose véritablement la question de la sélection des vues matérialisées qui doivent être créées. L'idée est de sélectionner les vues de manière à réduire le temps de réponse aux requêtes, tout en veillant à minimiser l'effort de maintenance et en tenant compte de ressources limitées, telles que l'espace de stockage alloué à ce type de structures. La difficulté, dans le cadre de systèmes analytiques, d'anticiper les requêtes qui vont être formulées par les utilisateurs, contribue grandement à la complexité de la tâche (Vaisman et Zimányi, 2014).

De nombreux algorithmes ont été développés, afin d'apporter une réponse au problème de sélection des vues matérialisées. Certains de ces algorithmes sont conçus de manière à parcourir l'ensemble des possibilités de matérialisation, et à identifier la meilleure alternative. D'autres fonctionnent avec des heuristiques qui permettent de réduire l'étendue des possibilités à explorer, mais qui ne garantissent pas d'aboutir à la solution optimale. Le principal argument en faveur du second type d'algorithmes est qu'en pratique, il est généralement inutile de perdre trop de temps à chercher la meilleure solution. L'essentiel est de trouver une solution qui correspond aux exigences des utilisateurs sans perdre trop de temps, et c'est ce que permettent de faire les heuristiques (Bouzeghoub, M. et Kedad, Z., 2000).

Les vues matérialisées ne délivrent leur vrai potentiel qu'une fois que des index sont définis sur elles, afin d'optimiser l'accès à leurs données. La méthodologie classique consiste à séparer la sélection des vues matérialisées et des index en deux phases distinctes. D'abord les vues sont choisies, et ensuite c'est au tour des index. Puisque les index et les vues matérialisées partagent souvent la même contrainte d'espace de stockage limité, qui doit donc être réparti entre les deux types de structures, cette approche peut s'avérer peu efficace. C'est la raison pour laquelle Gupta et al. (1997) considèrent que la sélection des vues matérialisées et celle des index doit se faire conjointement, en une seule et même étape.

Remarque : Les vues matérialisées qui contiennent des données agrégées sont souvent appelées des "Summary tables".

4.2 Les index

Un défi omniprésent dans le domaine des bases de données, est celui de l'accélération de l'accès aux données. C'est dans ce but que sont définis les index, qui sont sensés constituer un moyen rapide d'accéder aux données d'intérêt. Avec les vues matérialisées, les index constituent les deux principales méthodes d'optimisation des performances des systèmes de base de données (Golfarelli et Rizzi, 2009).

Tout comme pour les vues matérialisées, l'augmentation des performances du système au moyen d'index, présente des inconvénients sur le plan du stockage et de la maintenance (Vaisman et Zimányi, 2014). Le choix des index à créer doit donc être fait avec précaution, de manière à améliorer les performances en matière d'exécution des requêtes, sans trop réduire les performances en matière de stockage, et de mise à jour du data warehouse.

Certains auteurs considèrent que les particularités des systèmes OLAP, nécessitent l'utilisation d'index différents de ceux couramment utilisés dans le cadre des systèmes OLTP. Ces derniers étant pensés pour l'optimisation de requêtes opérationnelles, qui ne nécessitent l'accès qu'à un faible nombre de tuples, ils ne sont pas adaptés aux requêtes analytiques (Vaisman et Zimányi, 2014). De plus, le fait que les data warehouses soient des systèmes auxquels les utilisateurs accèdent uniquement en lecture, permet l'utilisation d'index dont la mise à jour peut être moins rapide que dans le cas des systèmes OLTP, sans que cela ne constitue un véritable problème (Golfarelli et Rizzi, 2009). Les index des data warehouses doivent aussi être en mesure de gérer efficacement la "sparsity" dans les données (Vaisman et Zimányi, 2014).

D’autres auteurs considèrent toutefois que, malgré la différence de nature entre les deux types de systèmes, des index populaires dans les systèmes OLTP (tels que les index B^+-tree), peuvent, dans certaines circonstances, s’avérer très utiles pour l’indexation des data warehouses (Golfarelli et Rizzi, 2009).

Deux types d’index courants pour les data warehouses sont les "bitmap indexes" et les "join indexes".

Il existe bien évidemment, d’autres types d’index utilisés dans les systèmes OLAP, comme par exemple les "projection indexes". Ces derniers se contentent de stocker dans une table la valeur d’un attribut pour l’ensemble des tuples d’une relation, sans modification de l’ordre ni suppression des doublons. La colonne correspondant à l’attribut sur lequel l’index est défini, est donc simplement dupliquée et stockée dans une autre table. Le raisonnement derrière ce type d’index est, simplement, qu’il est plus rapide de parcourir une structure simple (une liste) qu’une structure plus complexe (une table).

4.2.1 Bitmap indexes

Un "bitmap index" est un index qui prend la forme d’une matrice, avec autant de colonnes qu’il existe de valeurs distinctes pour l’attributs sur lequel l’index est construit, et autant de lignes que de tuples dans la table. Chaque case de cette matrice contient un bit, qui prend la valeur 1 lorsque l’attribut présente la valeur de la colonne pour une certaine ligne et 0 sinon. Ce type d’index permet donc d’identifier rapidement les tuples qui présentent une certaine valeur pour l’attribut indexé.

Le tableau ci-dessous représente un "bitmap index" sur l’attribut « couleur » d’une table « Produit ». Par exemple, le bit qui prend la valeur 1 dans la première ligne indique, que le produit 1 est de couleur rouge.

Tableau 4.1 : Bitmap index - Couleur des produits

Identifiant de ligne	Vert	Rouge	Jaune	Blanc	Noir
1	0	1	0	0	0
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	0	1	0

La performance de ce type d'index pour l'exécution des requêtes, vient de l'efficacité des opérations booléennes. Pour l'exécution d'opérations de type AND, OR ou NOT dans le cadre de conditions de sélection, il suffit de procéder à une simple comparaison de vecteurs de bits pour obtenir le vecteur résultant (Vaisman et Zimányi, 2014).

Étant donné que la taille de tels index est proportionnelle, à la fois au nombre de lignes, et au nombre de valeurs distinctes que peut prendre l'attribut indexé, ils conviennent surtout à l'indexation d'attributs à faible cardinalité. Autrement dit, aux attributs dont le domaine est composé d'un faible nombre de valeurs différentes. Pour ce type d'attributs, les "bitmap indexes" occupent moins d'espace en mémoire que beaucoup d'autres types d'index. Les utiliser avec des attributs à haute cardinalité implique, pour sa part, le stockage d'un grand nombre de vecteurs de bits peu denses, ce qui est loin d'être efficace ((Vaisman et Zimányi, 2014) ; (Golfarelli et Rizzi, 2009)).

Le fait que ces index soient stockés sous forme de vecteurs de bits peu denses, en fait de bons candidats à la compression. L'utilisation de méthodes de compression permet de résoudre d'éventuels problèmes d'efficacité de stockage, et par conséquent d'étendre les possibilités d'utilisation de ces index. Notons cependant, que le recours à des méthodes de compression implique des efforts de décompression (Vaisman et Zimányi, 2014).

Il existe des versions plus avancées des "bitmap indexes", telles que les "bit-sliced indexes" et les "bitmap-encoded indexes". Ils sont particulièrement utiles dans certains contextes, et souvent plus efficaces en matière d'espace de stockage, que les "bitmap indexes" classiques (Golfarelli et Rizzi, 2009). Ces indexes plus évolués ne sont cependant, pas développée ici.

4.2.2 Join indexes

Les "join indexes" sont des index utilisés afin d'accélérer l'exécution des jointures. Ces index sont particulièrement précieux dans les data warehouses, en raison de la structure en Star schema dans laquelle, la table de faits est reliée aux tables de dimensions par une multitude de clefs étrangères. La plupart des requêtes nécessitent en effet, la réalisation de jointures entre la table de faits et les tables de dimensions, sur base de ces clefs étrangères (Vaisman et Zimányi, 2014).

L'idée d'un "join index" est simplement de précalculer une jointure, et de stocker dans une table les paires d'identifiants des deux tables jointes, pour lesquels la condition de jointure est remplie. La réalisation de la jointure ne nécessite donc plus de scanner l'ensemble des deux

tables, mais uniquement de scanner l'index, ce qui résulte en un gain conséquent de performance.

Tout comme pour les "bitmap indexes", il existe des versions plus évoluées des "join indexes". On peut notamment citer les "star indexes", qui sont utilisés pour des jointures impliquant plus de deux tables, ou encore les "bitmap join indexes", qui stockent les résultats des jointures sous forme de matrices binaires.

4.2.3 La sélection des index

Comme pour les vues matérialisées, la sélection des index à mettre en place dans un data warehouse est un problème complexe, dans lequel beaucoup de facteurs interviennent : l'espace de stockage disponible, les caractéristiques du système de gestion de bases de données dans lequel la data warehouse est implémenté, l'anticipation des scénarios d'analyse et l'identification des attributs qui seront souvent impliqués dans les requêtes, etc. Et tout comme pour les vues matérialisées, de nombreux frameworks et algorithmes ont été développés afin d'aider les designers dans cette tâche.

De manière générale, l'indexation des tables de dimensions, qui sont surtout utilisées à des fins de sélection et de filtrage, se fait au moyen d'index de type bitmap ou B^+-tree (Golfarelli et Rizzi, 2009). Comme expliqué précédemment, les "bitmap indexes" sont plus adaptés aux attributs de faible cardinalité. Cependant, les méthodes de compression et les versions plus évoluées de ces index, réduisent les problèmes rencontrés en matière d'efficacité de stockage, avec les attributs qui n'entrent pas dans cette catégorie.

En ce qui concerne les tables de faits, il est souvent préférable de privilégier la création de multiples "join indexes" à celle d'un "star index", pour des raisons de flexibilité (Golfarelli et Rizzi, 2009).

4.3 Le partitionnement

Le partitionnement des données, aussi appelé fragmentation des données, consiste en la division du contenu des relations de la base de données en plusieurs sous-ensembles (des partitions), qui peuvent être traités de manière plus efficace. Les vues matérialisées, les index, aussi bien que les tables de base d'un datawarehouse, peuvent faire l'objet d'un partitionnement (Vaisman et Zimányi, 2014). L'attribut sur base duquel le partitionnement est réalisé, porte le nom de clef de partitionnement.

Une relation peut être partitionnée verticalement ou horizontalement. Dans le premier cas, la division se fait au niveau des colonnes. Les attributs sont donc répartis en groupes, stockés indépendamment les uns des autres. Une stratégie pour ce type de partitionnement peut être de rassembler, d'une part, les attributs les plus souvent utilisés, et d'autres part, ceux qui le sont moins. La logique derrière cette forme de partitionnement, est la même que celle précédemment évoquée pour les "projection indexes" (cf. section 4.2), à savoir qu'il est plus efficace d'utiliser des structures simples, que des structures complexes. En ce qui concerne le partitionnement horizontal, c'est au niveau des lignes que la division est réalisée. Les enregistrements de la table partitionnée sont répartis en sous-groupes, suivant un certain critère. Le temps est un critère souvent utilisé pour ce type de partitionnement dans les data warehouses (Vaisman et Zimányi, 2014).

Pour cette seconde forme de partitionnement, il existe différentes manières de répartir les tuples en plusieurs partitions. La façon de faire la plus courante, consiste à faire correspondre chaque partition à un intervalle de valeurs de la clef de partitionnement (Range partitioning). Une fonction de hashing, qui prend en entrée la valeur de la clef de partitionnement, peut aussi être utilisée afin de répartir les enregistrements de manière uniforme entre les différentes partitions (Hash partitioning). La répartition des lignes peut également être explicitement contrôlée, par la définition et l'assignement aux différentes partitions, de listes de valeurs de la clef de partitionnement (List partitioning).

Vaisman et Zimányi (2014) soulignent plusieurs avantages du recours au partitionnement. D'abord, la séparation des données des relations en plusieurs partitions permet l'amélioration des performances du système. En effet, d'une part, les possibilités de traitement en parallèle des différentes partitions d'une table réduisent grandement le temps nécessaire au traitement des données de cette dernière. D'autre part, le "partition pruning", qui consiste à n'utiliser que les partitions nécessaires à l'exécution d'une requête, plutôt que l'entièreté de la relation, permet de réduire encore davantage le temps de réponse, grâce à la réduction de la quantité de données à traiter. Pour les opérations de jointure sur la clef de partitionnement, joindre les partitions de deux tables est aussi globalement plus efficace, que de joindre ces deux tables en une seule fois. C'est d'autant plus vrai, en cas de traitement en parallèle des jointures entre partitions.

Ensuite, le partitionnement peut également avoir des retombées positives sur les tâches de maintenance, lorsque celles-ci ne doivent être réalisées que sur certaines parties d'une table.

Enfin, le partitionnement est utile dans le cadre d'architectures distribuées. Combiner le partitionnement des données, à leur distribution et leur réplication sur plusieurs serveurs, permet d'accroitre fortement la tolérance aux pannes d'un système. Cette propriété est très importante dans le cas des systèmes OLAP, puisqu'il est crucial d'assurer en permanence une bonne disponibilité des données pertinentes à la prise de décision et au monitoring des opérations. De plus, les données étant des données historisées, il n'est pas garanti qu'elles puissent être reconstituées en cas de défaillance totale du système, car il est probable qu'elles ne soient plus disponibles dans les systèmes sources.

Chapitre 5 : ETL

Dans ce chapitre, c'est la question du processus ETL qui est traitée. Les premières sous-sections sont dédiées à la définition du concept de processus ETL et à la présentation de ses différentes étapes. Le problème de la modélisation de ce type de processus est ensuite étudié. Enfin, les deux dernières parties du chapitre sont consacrées à la modélisation conceptuelle du processus ETL du data mart de l'étude de cas, et à son implémentation avec SQL Server Integration Services (SSIS).

5.1 Définition du concept d'ETL

"Les processus ETL (Extraction, Transformation, and Loading) sont utilisés afin d'extraire des données à partir de sources internes ou externes à une organisation, de transformer ces données, et de les charger dans un data warehouse." (Vaisman et Zimányi, 2014)

La conception du processus ETL peut facilement être considérée comme l'une des tâches les plus importantes lors de la mise en place d'un data warehouse. Il s'agit d'un travail complexe qui requiert une bonne maitrise des données, tout autant que des connaissances technologiques et liées à la gestion des processus. Il n'est pas rare que cette tâche soit celle à laquelle, la plus grande partie des ressources et des efforts de développement est consacrée (El-Sappagh, Hendawi et El Bastawissy, 2011).

5.2 Les étapes d'un processus ETL

La première étape d'un processus ETL, est l'étape d'extraction des données à partir des systèmes sources. La complexité de cette étape provient de l'hétérogénéité des systèmes et des plateformes qui les supportent. Chaque système présente généralement des particularités, qu'il faut être en mesure de gérer, afin de pouvoir correctement extraire les données qui devront être transformées et intégrées dans la suite du processus.

L'étape d'extraction connait deux phases distinctes durant la période d'utilisation d'un data warehouse : la phase d'extraction initiale, et la phase d'extraction incrémentale (El-Sappagh, Hendawi et El Bastawissy, 2011). L'extraction initiale a lieu une seule fois, lors de la première exécution du processus ETL. Le data warehouse, qui vient d'être créé, doit être chargé avec une grande quantité de données provenant des différents systèmes sources. L'extraction incrémentale, quant à elle, est réalisée à chacune des exécutions suivantes du processus ETL.

Les nouvelles données ainsi que les données qui ont subi des modifications depuis la dernière extraction, sont extraites des systèmes sources pour pouvoir rafraichir le contenu du data warehouse.

La deuxième étape d'un processus ETL, est l'étape de transformation. L'objectif durant cette partie du processus est d'appliquer aux données extraites, les transformations nécessaires afin d'obtenir un ensemble de données complet, correct, cohérent, et dépourvu d'ambiguïté. Les données issues des différents systèmes sous-jacents sont ici nettoyées, transformées et intégrées (El-Sappagh, Hendawi et El Bastawissy, 2011).

La troisième et dernière étape d'un processus ETL, l'étape de chargement, consiste à récupérer les données qui ont été extraites et transformées, pour les introduire dans les tables de dimensions et de faits du data warehouse, et ainsi les mettre à disposition des utilisateurs.

5.3 Le problème de la modélisation des processus ETL

La qualité du design du processus ETL et la capacité à le maintenir efficacement, sont deux facteurs d'une importance capitale, pour la réussite d'un projet de data warehousing (Sonal et Rajni, 2014). Malgré cela, une grande partie de la littérature sur le sujet déplore l'absence d'un modèle standard pour le design de tels processus. Non seulement, chaque outil consacré à la phase d'ETL propose sa propre manière de définir les processus et son propre langage, mais aucun modèle conceptuel ne semble parvenir à s'imposer non plus.

De manière générale, les propositions de la littérature concernant la modélisation des processus ETL peuvent être regroupées en trois grandes catégories : les modèles basés sur les expressions de mapping et les directives, les modèles basés sur des construits conceptuels propres à l'ETL, et les modèles basés sur UML ((Sonal et Rajni, 2014) ; (El-Sappagh, Hendawi et El Bastawissy, 2011)).

En ce qui concerne le premier type de modèles, les expressions de mapping entre les systèmes sources et le data warehouse, prennent la forme de requêtes exécutées par le système de gestion de bases de données. C'est ce dernier qui assume donc le rôle d'outil d'ETL. Les directives en complément des expressions de mapping, sont pour leur part généralement définies manuellement et conservées par écrit. La difficulté à maintenir les processus ETL ainsi conçus, est le grand inconvénient qui accompagne cette façon de procéder. Il est en effet très compliqué d'implémenter des changements, puisque cela passe par une modification manuelle des requêtes et des directives (Sonal et Rajni, 2014).

Les modèles qui se basent sur des construits conceptuels ont quant à eux, l'avantage de reposer sur une notation graphique, qui permet la modélisation explicite de toutes les activités pouvant intervenir dans le cadre d'un processus ETL. Une telle approche permet de générer des processus complets, détaillés, ainsi que simples à maintenir et à étendre (Sonal et Rajni, 2014). Un exemple d'un tel modèle est le modèle "Entity Mapping Diagram" (EMD) proposé par El-Sappagh, Hendawi et El Bastawissy (2011), pour le design conceptuel des processus ETL. Ce modèle conceptuel, conçu spécialement pour le design des processus ETL, repose sur une notation qui permet, selon ses créateurs, de représenter de manière simple les étapes d'extraction, de transformation et de chargement des données.

La troisième famille de modèles, qui s'appuie sur UML pour modéliser les processus ETL, présente pour premier avantage de ramener les processus ETL complexes, à un ensemble de tâches simples. Cependant, cet atout peut rapidement se transformer en inconvénient, si la simplification est poussée trop loin. En effet, une simplification excessive peut rendre ce type de modèles, incapable de représenter certains des concepts pertinents dans le cadre d'un processus ETL. En ce sens, ils peuvent être considérés comme moins adéquats pour la réalisation d'une modélisation complète et correcte, que le deuxième type de modèles présenté ci-dessus (Sonal et Rajni, 2014).

Trujillo et Luján-Mora (2003), qui ont proposé un modèle de ce type, mettent en avant le fait qu'il s'agisse d'une notation standard, maitrisée par beaucoup de designers. Par conséquent, son utilisation permet de supprimer les efforts, qui devraient être consentis afin de se familiariser avec une nouvelle notation. Ils ajoutent que le recours à un tel modèle peut s'inscrire dans une approche plus globale du projet de data warehousing, lorsqu'il est combiné à un design conceptuel du data warehouse qui repose également sur UML. Un seul langage est ainsi utilisé, pour l'ensemble des démarches de design conceptuel des différents aspects du projet. Les deux auteurs décrivent le modèle qu'ils proposent, et pour lequel UML est utilisé afin de représenter les principales tâches des processus ETL, comme une approche simple, qui facilite le design et la maintenance de ces processus.

5.4 Une approche pour la modélisation conceptuelle des processus ETL **(Vaisman et Zimányi, 2014)**

Selon Vaisman et Zimányi (2014), étant donnée la complexité inhérente aux processus ETL, il est crucial de parvenir à réduire les efforts de développement et de maintenance de ces derniers. Le passage par une étape de modélisation conceptuelle de ces processus, est un moyen

efficace d'atteindre cet objectif. En effet, comme c'est le cas pour le design conceptuel d'un data warehouse (cf. section 2.1), il est pertinent, pour les processus ETL, de recourir à une modélisation indépendante de toute plateforme d'implémentation, de manière à cacher les détails techniques. Cela offre la possibilité aux utilisateurs et aux designers de se focaliser sur les éléments centraux du processus. De plus, les modèles ainsi générés peuvent, par la suite, être utilisés avec n'importe quelle plateforme d'implémentation, et demeurent valables en cas de passage d'une plateforme à une autre.

Pour la modélisation conceptuelle des processus ETL, les deux auteurs proposent d'utiliser la "Business Process Modeling Notation" (BPMN), qui est une notation graphique utilisée afin de définir et modéliser les processus métiers. Selon eux, BPMN fournit les éléments nécessaires à la modélisation des processus ETL les plus fréquemment utilisés. Puisque BPMN est défini sur base d'UML, cette approche peut être considérée comme faisant partie de la troisième famille de modèles présentée dans la sous-section précédente.

Cette notation dispose de l'avantage d'être un standard dans le milieu de la modélisation de processus, et d'avoir été conçue de manière à pouvoir être utilisée aussi bien par l'IT que par le business. Il s'agit donc d'une notation abordable pour beaucoup d'utilisateurs, et avec laquelle beaucoup de designers sont familiers.

L'approche ici proposée, représente les processus ETL sous-forme d'une combinaison de deux types d'éléments : les "control tasks" et les "data tasks". Ils peuvent être compris comme deux manières différentes de visualiser un processus ETL.

Les "control tasks" et leurs sous-tâches, sont utilisées afin de modéliser l'orchestration du processus. Une "control task" représente généralement le chargement du data warehouse, et chaque sous-tâche représente une des grandes activités de ce chargement. Les liens qui unissent les différentes tâches, permettent ici uniquement d'établir des relations de précédence entre ces dernières.

Les "data tasks", pour leur part, permettent de représenter les activités de manipulation des données réalisées durant le processus. Les relations entre ces tâches ne symbolisent donc pas uniquement un lien de précédence, mais représentent aussi des flux de données. Une "data task" est dite unaire, lorsqu'elle ne prend qu'un seul flux de données en entrée, et n-aire, lorsqu'elle prend n flux de données en entrée. Une distinction peut aussi être réalisée, entre les tâches qui réalisent les manipulations et transformations ligne par ligne (row operations) et celles qui traitent les lignes par ensembles (rowset operations).

L'annexe 2 contient la liste des "data tasks" qui peuvent être utilisées afin de modéliser un processus ETL. Chaque tâche est accompagnée d'annotations qui contiennent des métadonnées pertinentes la concernant.

Puisque chaque "control task" renferme une série de "data tasks", ces dernières peuvent être considérées comme représentant le processus à un niveau d'abstraction inférieur. Alors que les "control tasks" donnent une vue globale du processus ETL et des grandes activités qui doivent être réalisées, les "data tasks" s'intéressent en détails aux traitements subis par les données.

5.5 Modélisation conceptuelle du processus ETL de l'étude de cas

Pour la modélisation conceptuelle du data mart de l'étude de cas, c'est l'approche proposée par Vaisman et Zimányi (2014) qui est suivie. La modélisation se fait donc en deux temps, sur base de BPMN. D'abord, une "control task" et ses sous-tâches fournissent une vue globale de l'orchestration du processus. Ensuite, les manipulations de données sont détaillées au travers de "data tasks".

5.5.1 Control task

Le chargement du data mart et celui des tables qui le composent, constituent respectivement la "control task" du processus ETL et ses sous-tâches. Il a ici été décidé, pour des raisons de lisibilité des schémas, de scinder la "control task" du chargement du data mart en deux "contol tasks" distinctes. Chacun des deux schémas correspond ainsi au chargement du data mart pour un des deux types de ventes. Le fait que la plupart des sous-tâches apparaissent dans les deux "control tasks", ne signifie toutefois pas que le chargement doit être effectué une fois par type de ventes. Chaque table n'est chargée qu'une seule fois au cours du processus.

Sur ces deux schémas, les relations de précédence qui définissent l'ordre dans lequel les actions doivent être exécutées, dépendent des contraintes d'intégrité référentielles qui unissent les différentes tables. En effet, il est nécessaire qu'une table référencée par une autre au moyen d'une clef étrangère, soit chargée dans le data mart avant la table qui la référence. (ex : la table Dim_ProductCategory doit être chargée avant la table Dim_ProductSubcategory, qui doit elle-même être chargée avant la table Dim_Product). C'est la raison pour laquelle les tables de faits sont les dernières tables à être chargées.

En ce qui concerne l'utilisation des gateways parallèles, celles de séparation signifient simplement que tous les chemins sortants doivent être suivis et donc, que toutes les tâches sur ces chemins doivent être exécutées. Les gateways de jointure expriment pour leur part, qu'il est nécessaire d'attendre que toutes les tâches sur les chemins entrants soient terminées avant de pouvoir continuer le processus.

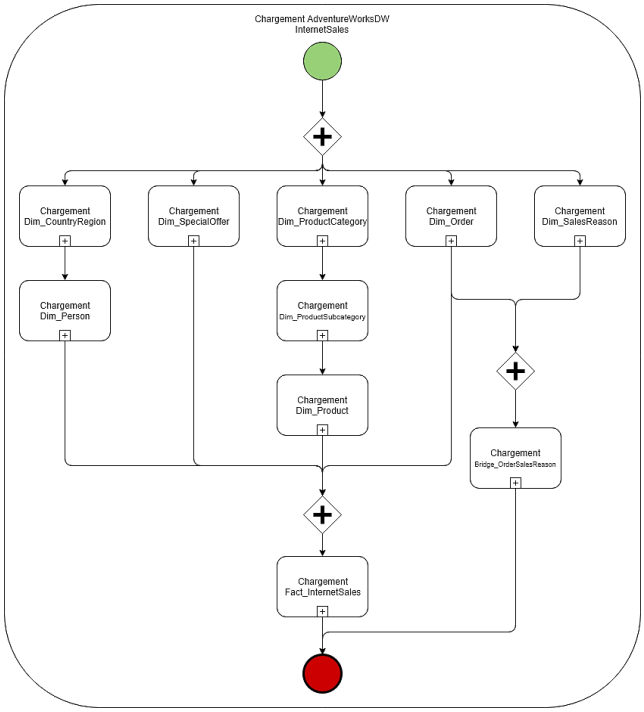


Figure 5.1 : Control task - InternetSales

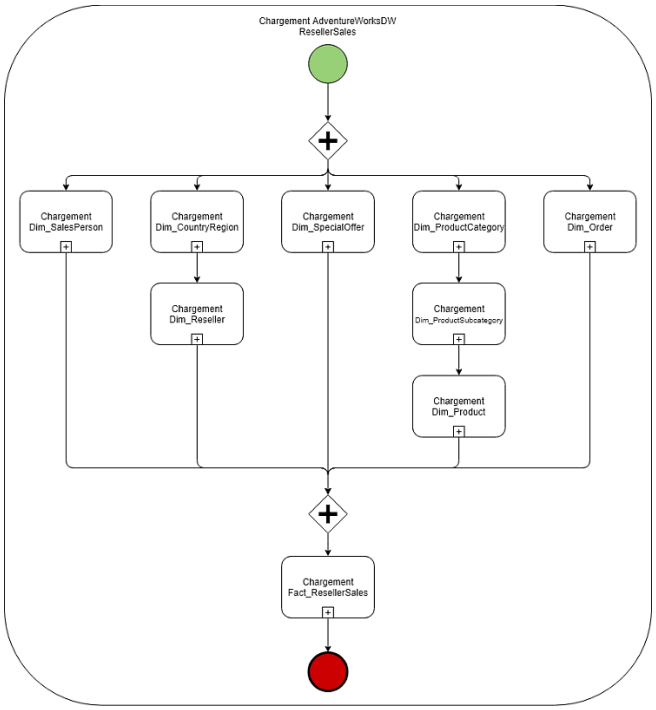


Figure 5.2 : Control task - ResellerSales

Remarque : Le chargement de la dimension temporelle n'est pas mentionné sur les schémas, car cette dimension est générée avec SQL Server Analysis Services (cf. section 6.1.2).

5.5.2 Data tasks

Les différents schémas qui suivent, représentent le détail des traitements subis par les données au cours du processus, de leur extraction depuis le système source, jusqu'à leur chargement dans les tables préalablement créées du data mart. Chaque schéma correspond aux détails du chargement d'une table en particulier.

Dans ces schémas, plusieurs types de tâches sont utilisés. Les tâches "Input Data" précisent l'origine des données manipulées : une table du système source, ou une requête exécutée sur ce dernier. Les tâches "Add Colmumn" indiquent qu'une colonne supplémentaire est ajoutée au flux de données, ainsi que la manière dont son contenu est obtenu. Ces tâches sont, entre autres, utilisées afin d'ajouter, dans le cas des tables de dimensions, des colonnes vides destinées à la génération de "surrogate keys". Les tâches "Update Column" symbolisent une modification apportée au contenu d'un champ du flux. Les tâches "Lookup" représentent l'action de récupérer le contenu d'un champ d'une table ou d'une requête, selon certaines conditions. Ce type de tâches est notamment utilisé afin de récupérer les valeurs des clefs primaires d'autres tables, en vue de la création de clefs étrangères vers ces dernières. Enfin, les tâches "Insert Data" symbolisent l'insertion des données transformées dans les tables du data mart. Ce dernier type de tâches, s'accompagne d'un mapping entre les champs du flux de données et les champs de la table d'insertion. Les champs du flux de données qui n'apparaissent pas dans ce mapping sont simplement considérés comme étant supprimés du flux.

Chargement des tables de dimensions :

En ce qui concerne la table Dim_CountryRegion (cf. figure 5.3), la seule manipulation qui est réalisée avant l'insertion des données dans le data mart, est l'ajout d'une colonne vide dans laquelle la "surrogate key" est générée au moment de cette insertion. C'est également le cas pour le chargement des tables Dim_ProductCategory (cf. figure 5.6), Dim_Order (cf. figure 5.9), Dim_SalesReason (cf. figure 5.10), Dim_SalesPerson (cf. figure 5.12) et Dim_SpecialOffer (cf. figure 5.13).

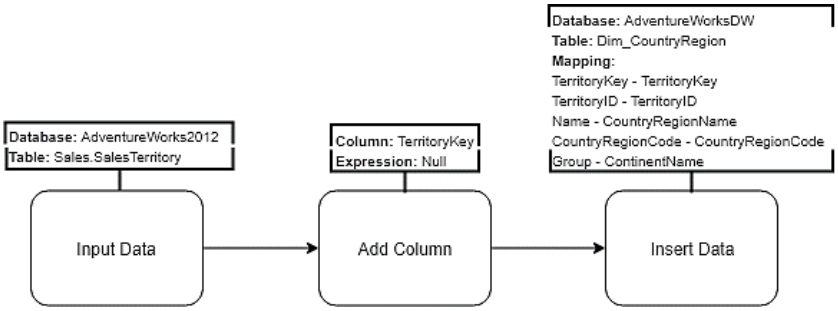


Figure 5.3 : Data tasks - Chargement Dim_CountryRegion

Pour les tables Dim_Person (cf. figure 5.4) et Dim_reseller (cf. figure 5.5), en plus de la colonne vide destinée à la génération de la "surrogate key", des colonnes sont créées afin d'accueillir les informations extraites du document XML contenu dans le champ "Demographics". Le chargement de ces deux tables requiert aussi la récupération de la clef primaire de la table Dim_CountryRegion, puisqu'elles référencent toutes deux cette dernière, via une clef étrangère.

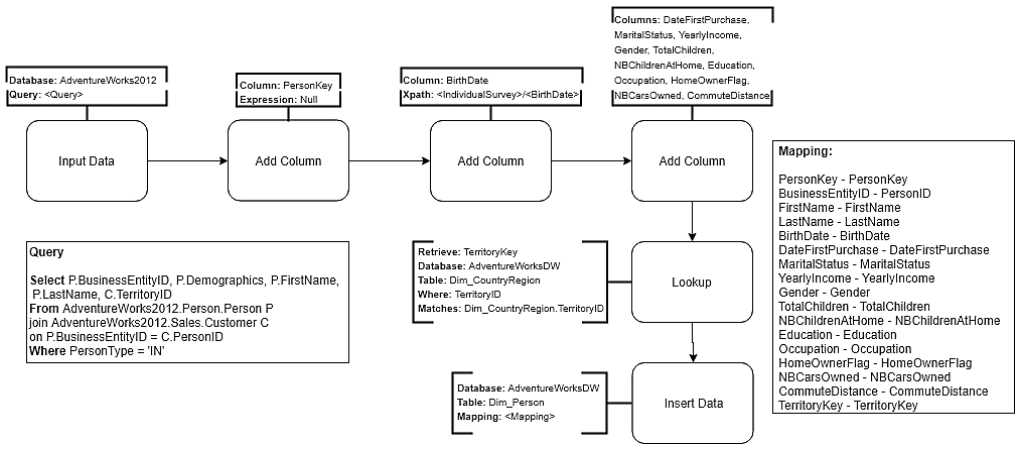


Figure 5.4 : Data tasks - Chargement Dim_Person

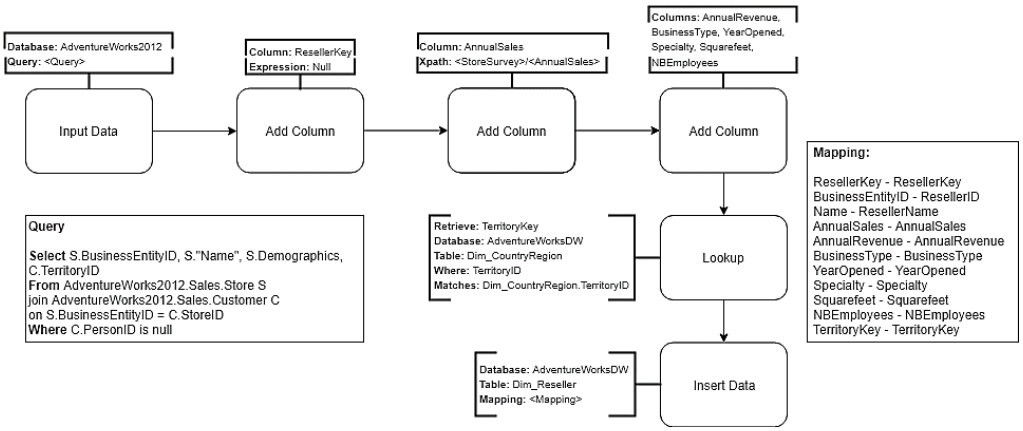


Figure 5.5 : Data tasks - Chargement Dim_Reseller

Au niveau des tables Dim_ProductSubcategory (cf. figure 5.7) et Dim_Product (cf. figure 5.8), ce sont respectivement, la clef primaire de table Dim_ProductCategory et celle de la table Dim_ProductSubcategory, qui sont récupérées avec des tâches "Lookup", en vue de la mise en place de clefs étrangères.

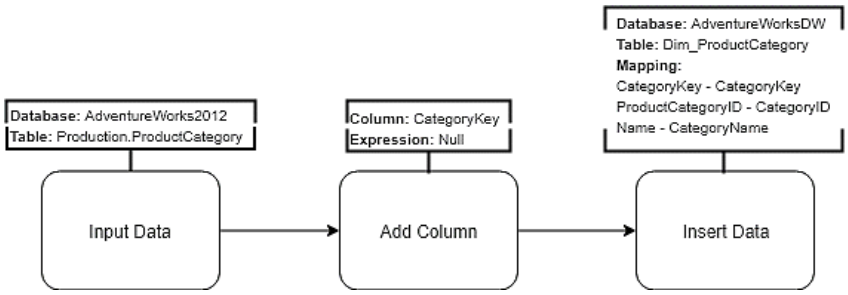


Figure 5.6 : Data tasks - Chargement Dim_ProductCategory

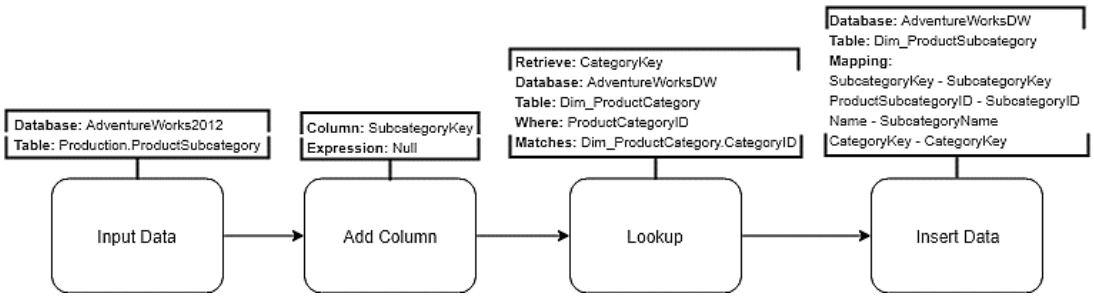


Figure 5.7 : Data tasks - Chargement Dim_ProductSubcategory

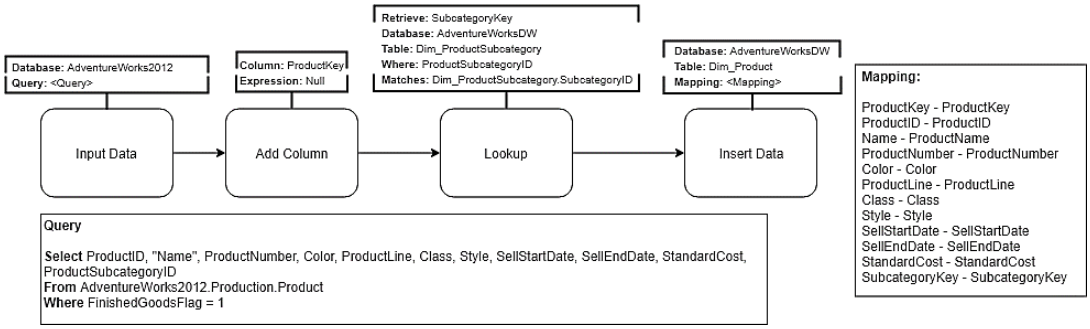


Figure 5.8 : Data tasks - Chargement Dim_Product

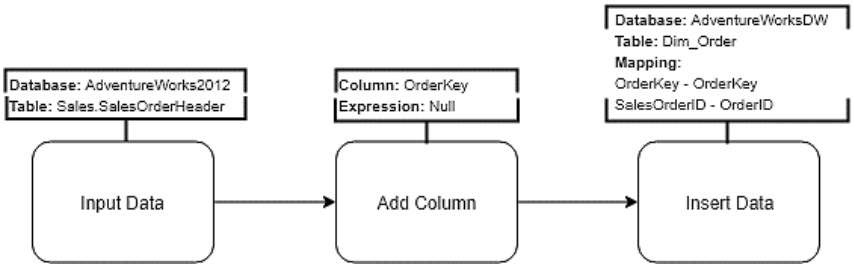


Figure 5.9 : Data tasks - Chargement Dim_Order

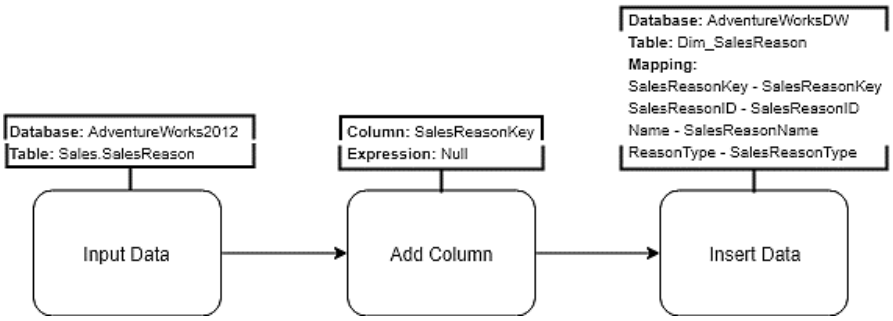


Figure 5.10 : Data tasks - Chargement Dim_SalesReason

La première tâche "Lookup" qui intervient dans le chargement de la table Bridge_OrderSalesReason (cf. figure 5.11), permet de récupérer à partir d'une requête SQL, le nombre de raisons d'achat associées à chaque commande. Cette information est ensuite utilisée afin de calculer la valeur de la colonne "Percentage". Après l'ajout de cette colonne, les tâches suivantes consistent à récupérer les clefs primaires des tables Dim_order et Dim_SalesReason, et à insérer les données dans le data mart.

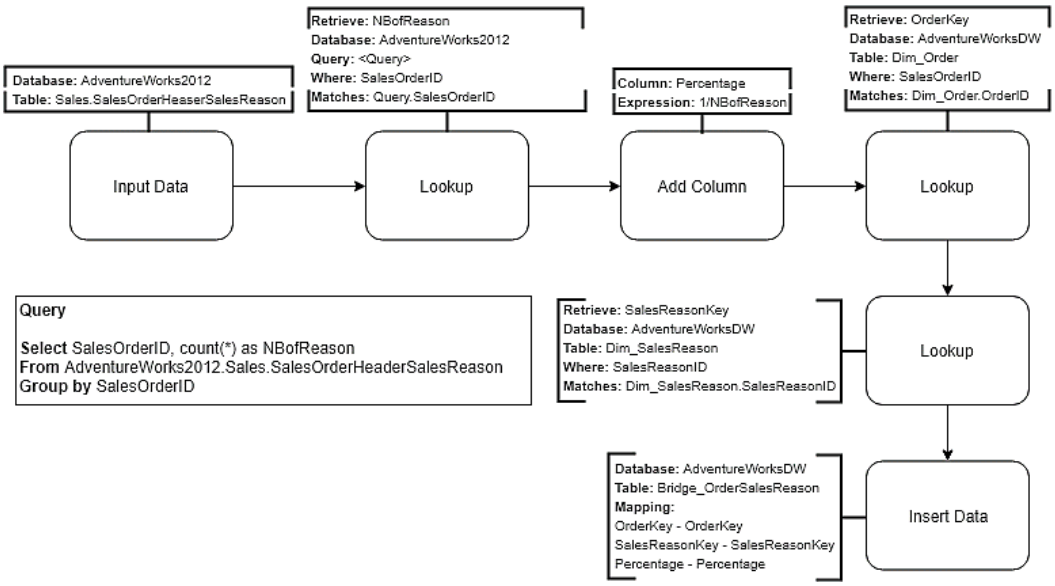


Figure 5.11 : Data tasks - Chargement Bridge_OrderSalesReason

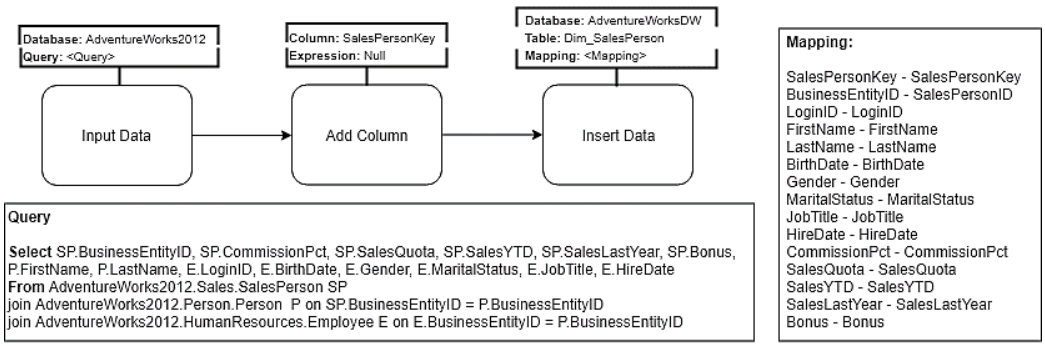


Figure 5.12 : Data tasks - Chargement Dim_SalesPerson

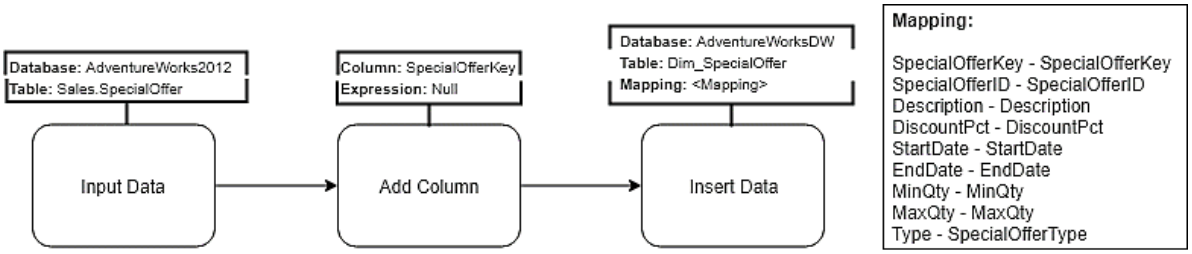


Figure 5.13 : Data tasks - Chargement Dim_SpecialOffer

Chargement des tables de faits :

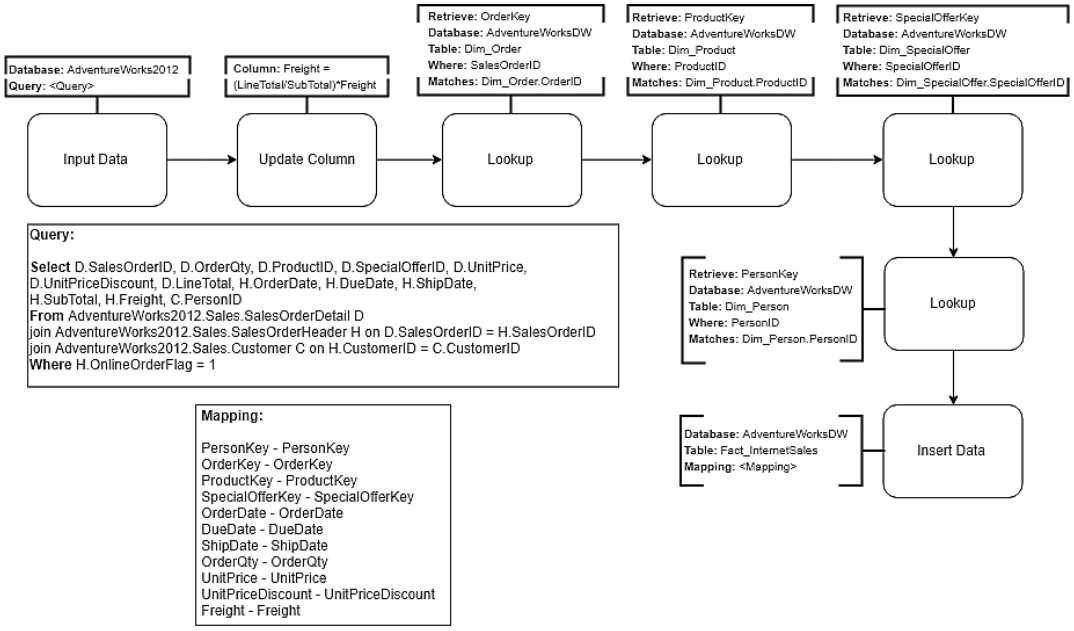


Figure 5.14 : Data tasks - Chargement Fact_InternetSales

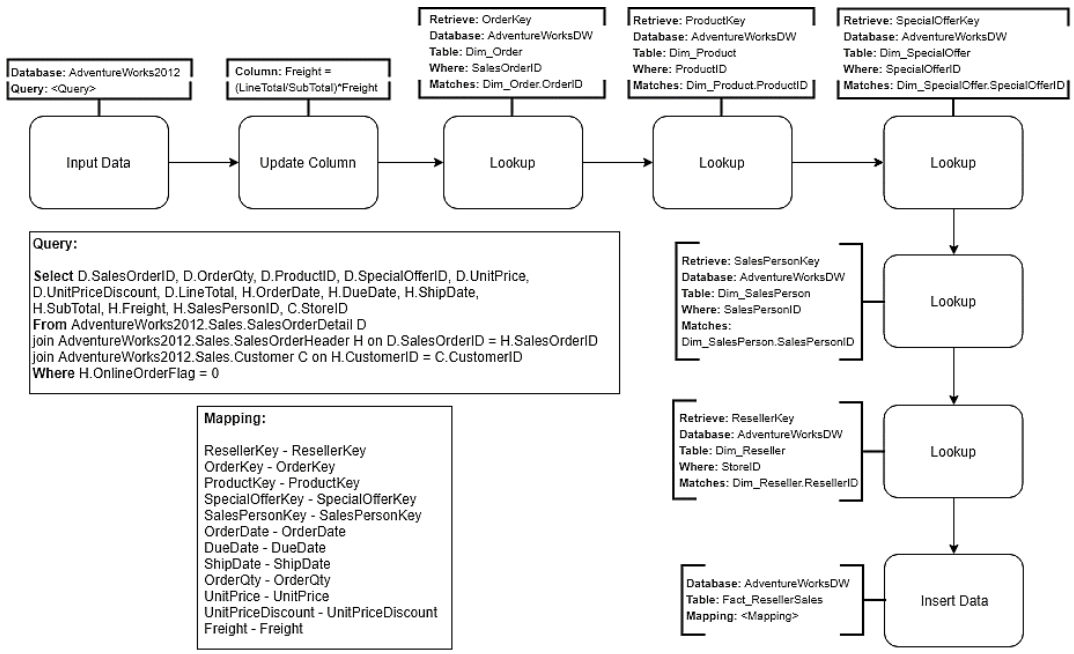


Figure 5.15 : Data tasks - Chargement Fact_ResellerSales

En ce qui concerne les deux tables de faits, la première étape du traitement des données, après leur extraction depuis la base de données source, est la mise à jour du champ "Freight". Cette dernière vise à répartir les frais de livraison de chaque commande entre les différentes ventes qui lui sont associées, proportionnellement à leur importance dans la commande. Les clefs primaires des différentes tables de dimensions sont ensuite récupérées, avant d'insérer les données dans les tables.

5.6 Implémentation du processus ETL de l'étude de cas avec SQL Server Integration Services

Integration Services est l'outil de migration de données de la suite SQL Server. Un package SSIS contient un flux de contrôle ainsi qu'une série de flux de données.

5.6.1 Flux de contrôle

Le flux de contrôle est constitué d'un ensemble de tâches reliées les unes aux autres par des contraintes de précédence, qui déterminent leur ordre d'exécution. Il fournit une vue globale de l'orchestration du processus ETL, et peut donc être perçu comme l'équivalent de la "control task" du modèle conceptuel.

Tout comme pour la "control task", les différentes tâches du flux de contrôle correspondent au chargement des différentes tables du data mart. Les contraintes de précédence qui unissent ces tâches, sont définies conformément aux contraintes d'intégrité référentielle qui lient les tables du data mart.

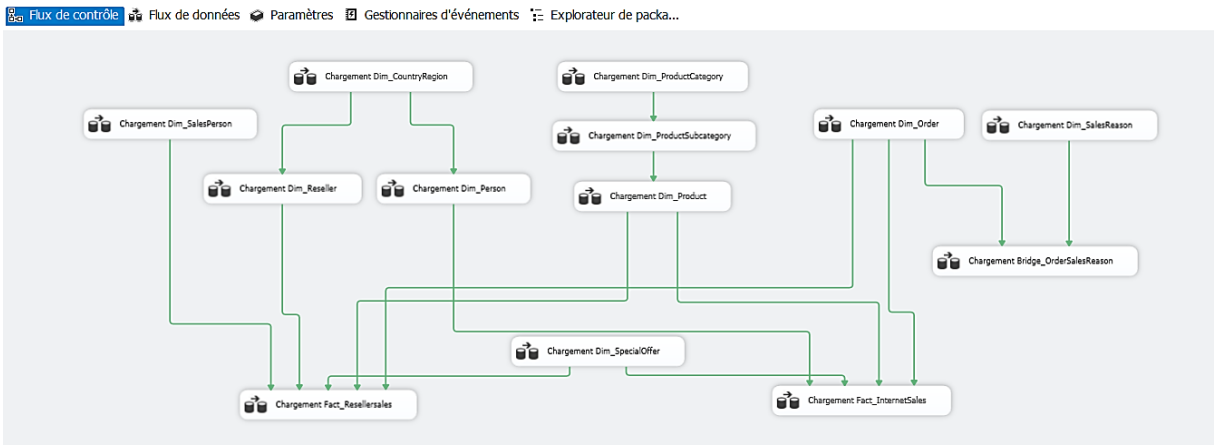


Figure 5.16 : Flux de contrôle du chargement du data mart

5.6.2 Flux de données

Un flux de données est défini pour chacune des tâches du flux de contrôle. Ces flux, de manière similaire aux "data tasks" du modèle conceptuel, définissent les traitements subis par

les données durant le processus. Trois types d'éléments sont fournis par l'outil, pour la création de flux de données : les sources, qui permettent d'extraire les données qui doivent être migrées, les transformations, grâce auxquelles des modifications peuvent être apportées à ces données, et les destinations, qui permettent de les charger dans les structures prévues afin de les accueillir, une fois le processus achevé.

Pour le chargement de chacune des tables du data mart, le flux de données débute par une source, qui établit la connexion avec la base de données opérationnelle de laquelle les données sont extraites. De la même manière, chacun des flux se termine par une destination, qui établit la connexion avec le data mart, dans lequel les données doivent être chargées.

Chargement des tables de dimensions :

Pour le chargement de la table Dim_CountryRegion (cf. figure 5.17), aucune transformation n'est nécessaire. Les données sont simplement extraites du système source et introduites dans le data mart.

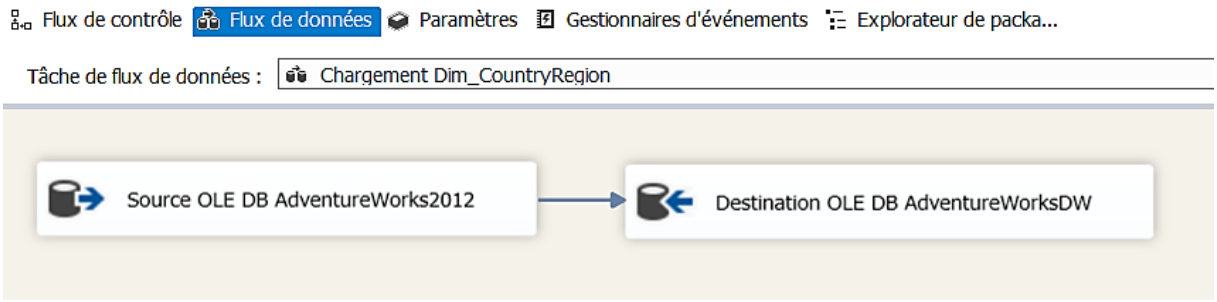


Figure 5.17 : Flux de données - Chargement Dim_CountryRegion

Le chargement des tables Dim_ProductCategory (cf. figure 5.22), Dim_Order (cf. figure 5.25), Dim_SalesReason (cf. figure 5.26), Dim_SalesPerson (cf. figure 5.28) et Dim_SpecialOffer (cf. figure 5.29) suit le même processus, dépourvu de transformations.

Le mapping réalisé au niveau de la destination, entre les champs du flux et ceux de la table d'insertion Dim_CountryRegion, est fourni à la figure 5.18. Dans ce mapping, la colonne TerritoryKey de la table d'insertion est ignorée car il s'agit d'une colonne auto-générée. Pour toutes les tables de dimensions, la génération des "surrogate keys" est gérée par SQL Server au moment de l'insertion, via ce mécanisme.

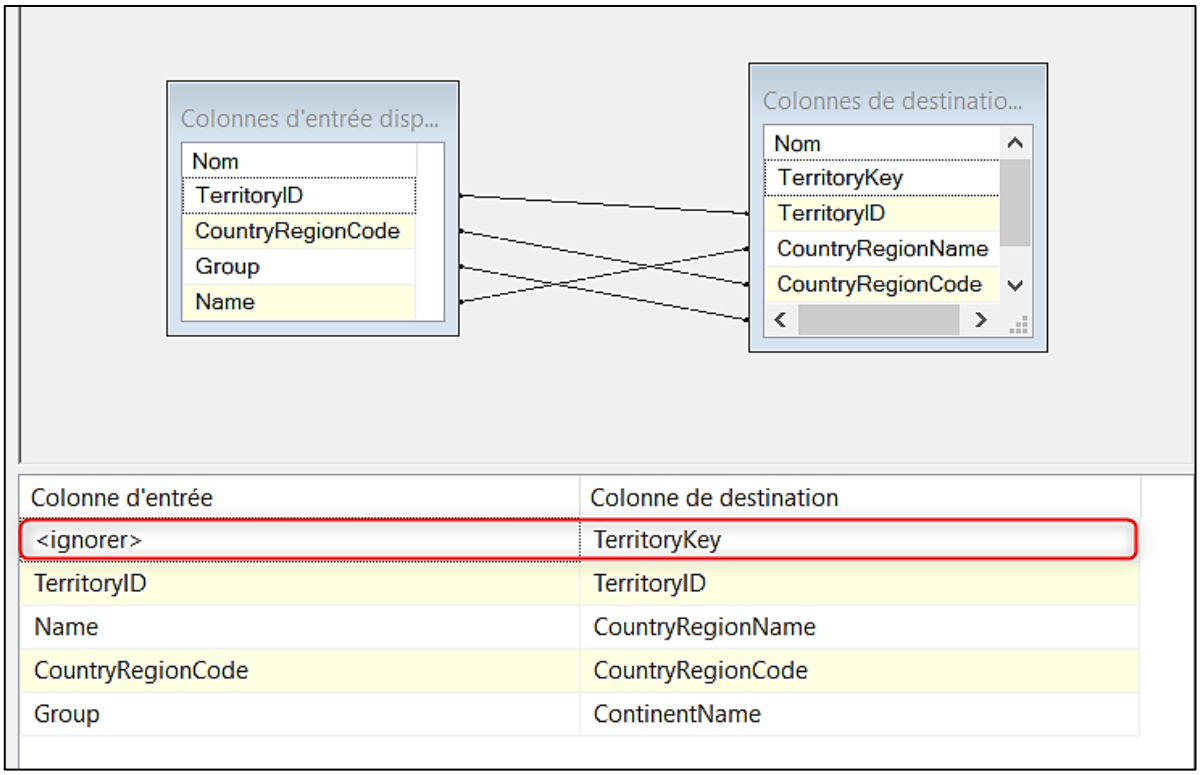


Figure 5.18 : Mapping - Dim_CountryRegion

Pour le chargement des tables Dim_Person (cf. figure 5.19) et Dim_Reseller (cf. figure 5.20), le flux de données passe par une transformation de recherche (Lookup). Cette transformation permet d'ajouter à ce dernier, le champ "TerritoryKey", récupéré dans la table Dim_CountryRegion à l'aide d'une jointure sur l'attribut "TerritoryID" (cf. figure 5.21). Cet attribut est récupéré en vue de la création d'une clef étrangère. Une transformation similaire est utilisée à chaque fois qu'un système de clef étrangère doit être mis en place entre deux tables.

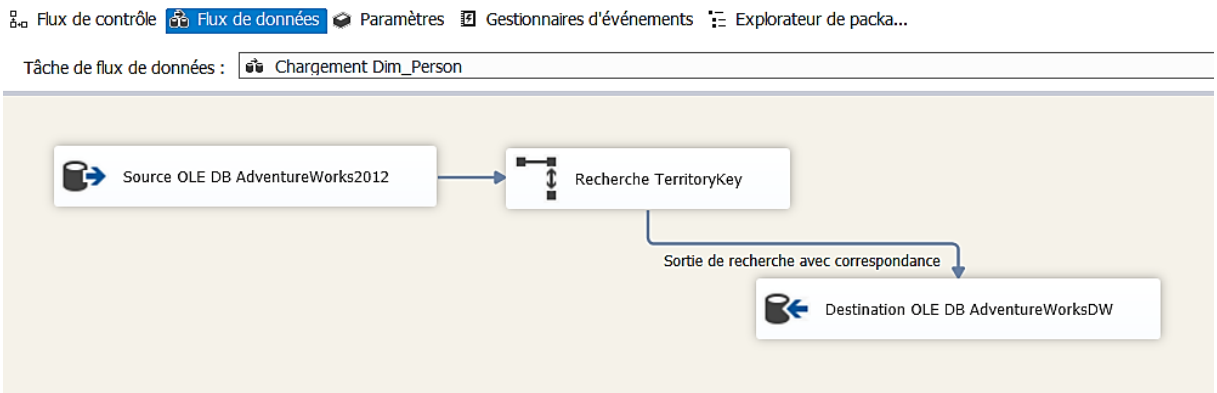


Figure 5.19 : Flux de données - Dim_Person

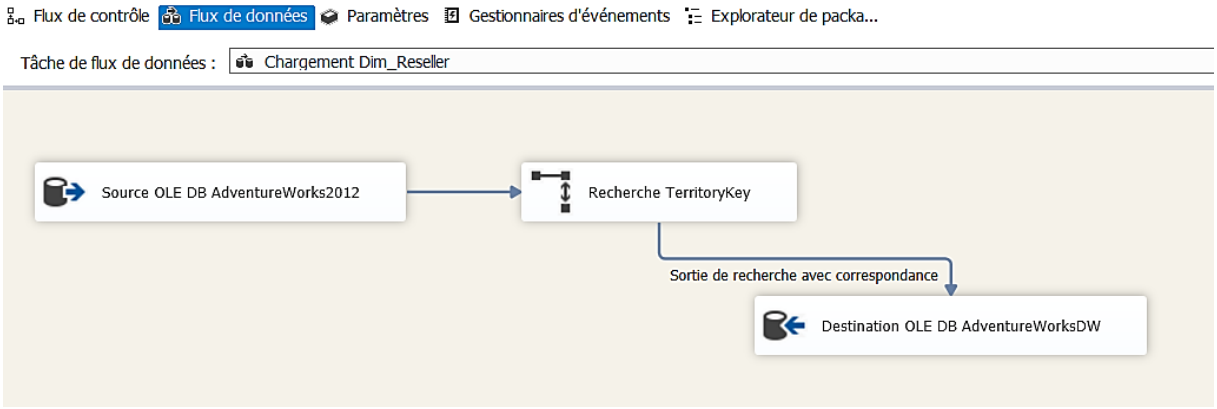


Figure 5.20 : Flux de données - Dim_Reseller

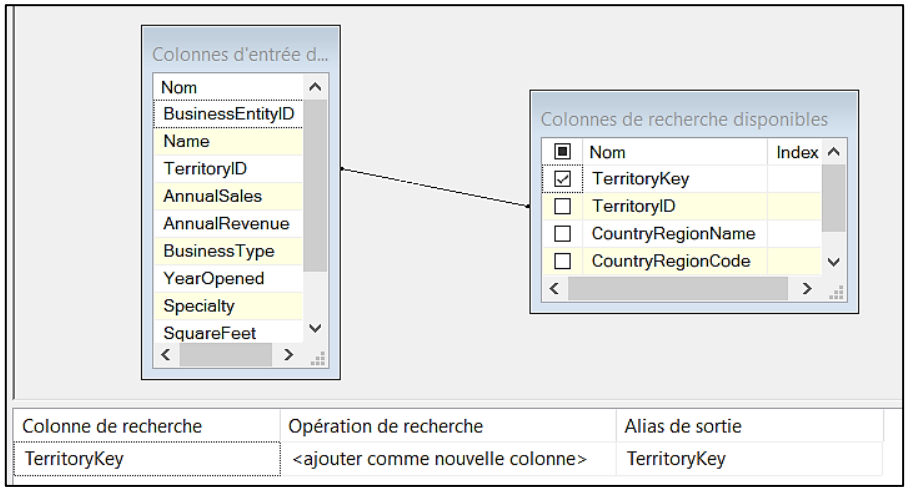


Figure 5.21 : Recherche TerritoryKey - Dim_Reseller

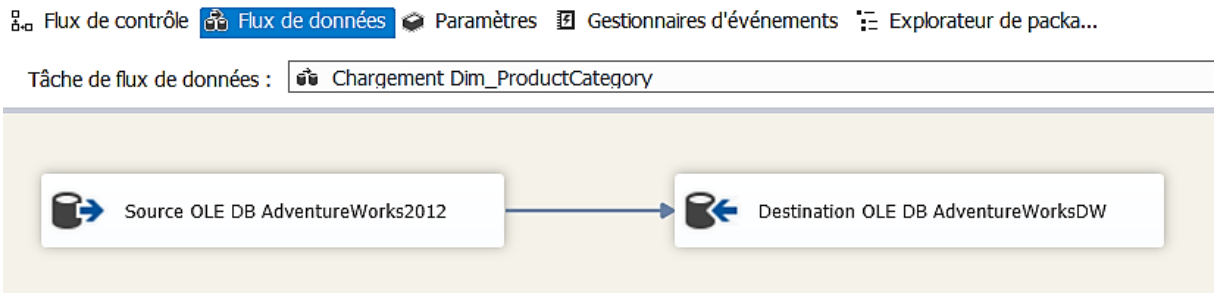


Figure 5.22 : Flux de donnée - Dim_ProductCategory

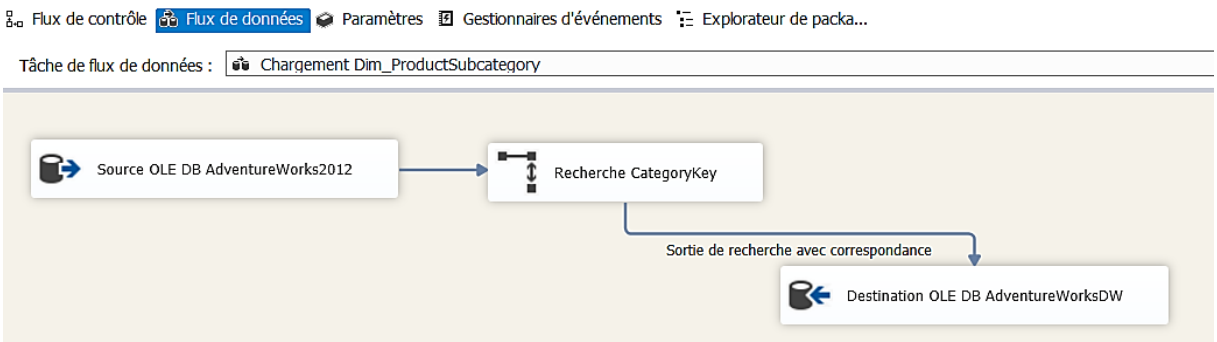


Figure 5.23 : Flux de données - Dim_ProductSubcategory

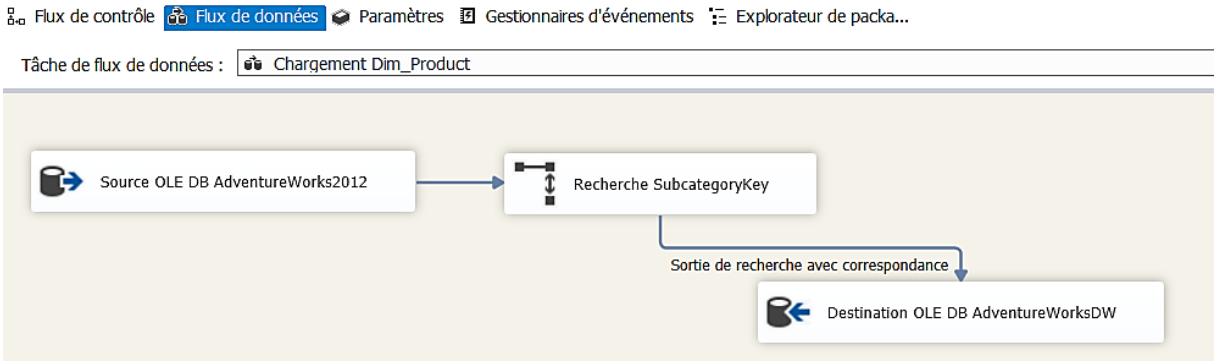


Figure 5.24 : Flux de données - Dim_Product

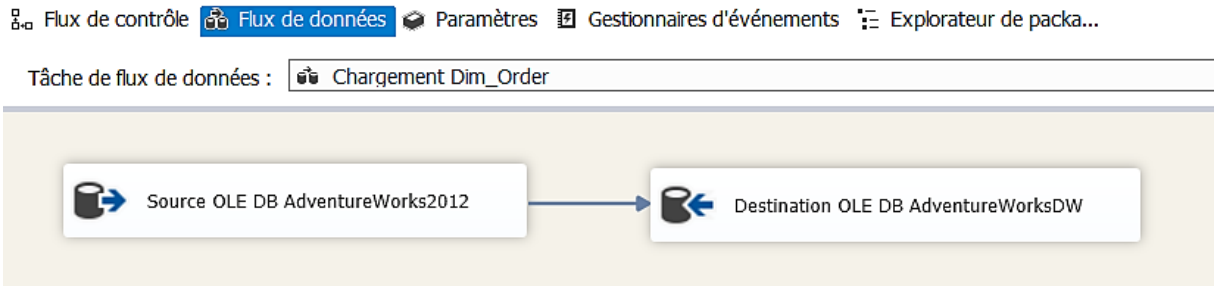


Figure 5.25 : Flux de données - Dim_Order

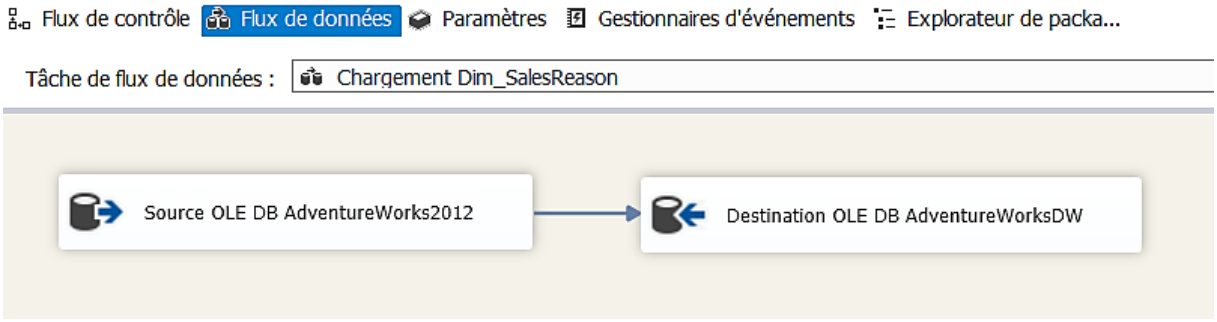


Figure 5.26 : Flux de données - Dim_SalesReason

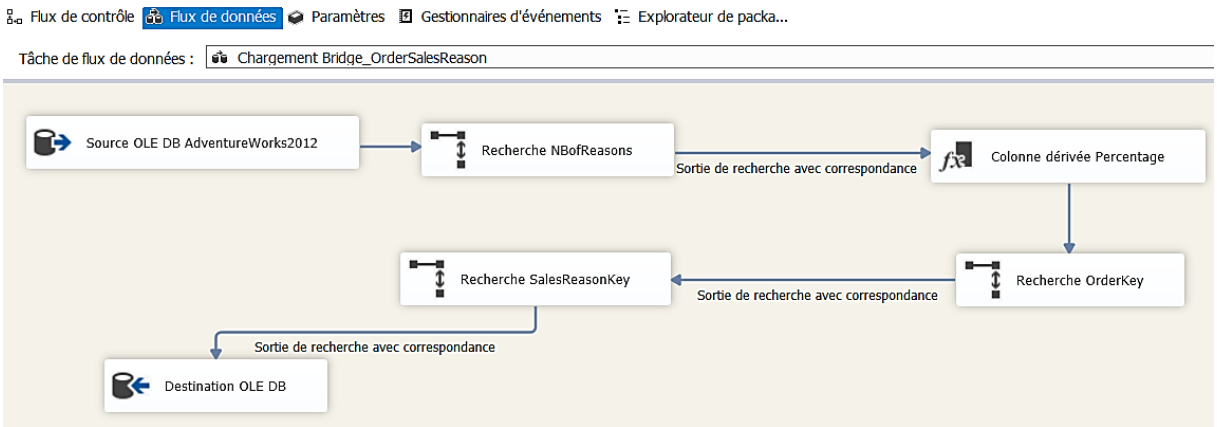


Figure 5.27 : Flux de données - Bridge_OrderSalesReason

La première transformation de recherche qui intervient lors du chargement de la table Bridge_OrderSalesReason (cf. figure 5.27), permet d'ajouter au flux, le nombre de raisons

d'achat associées à chaque commande. Cette information, obtenue grâce à une jointure avec le résultat d'une requête SQL, intervient ensuite directement dans le calcul de la colonne "Percentage".

$$Percentage = \frac{1}{NBofReasons}$$

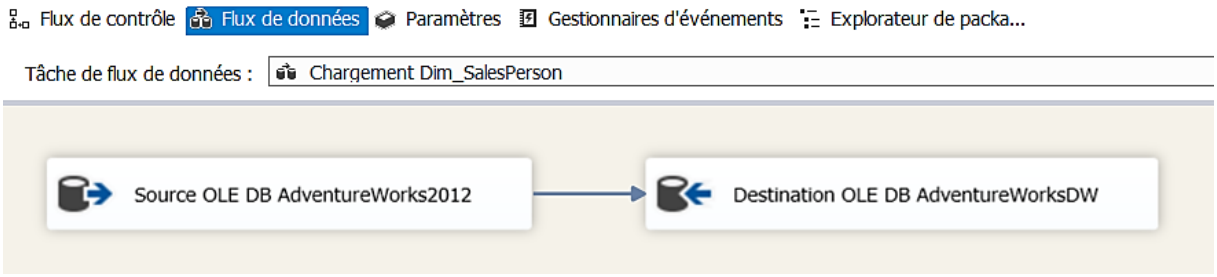


Figure 5.28 : Flux de données - Dim_SalesPerson

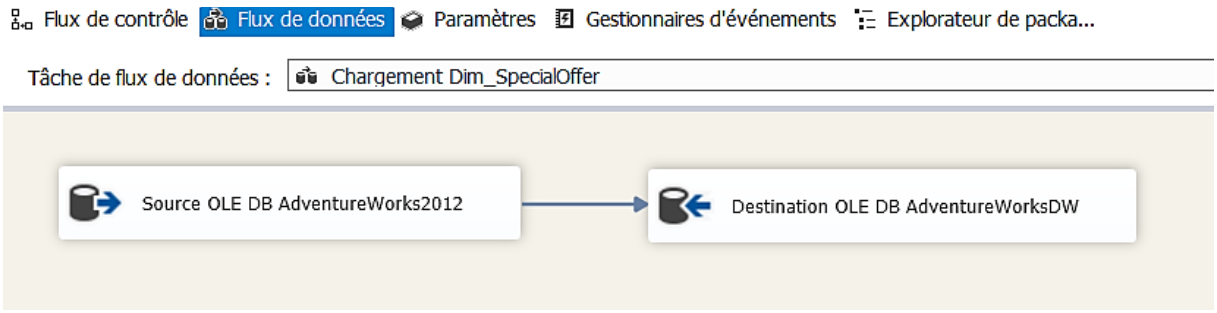


Figure 5.29 : Flux de données - Dim_SpecialOffer

Chargement des tables de faits :

Concernant le chargement des deux tables de faits, la première transformation qui est réalisée, est l'ajout d'une colonne dérivée dans laquelle les frais de livraison de chaque commande sont répartis entre les différentes ventes qui lui sont associées. La répartition est faite conformément la formule suivante :

$$Freight = \frac{LineTotal}{SubTotal} * Freight$$

Les transformations suivantes permettent d'ajouter aux flux de données, les clefs primaires des différentes tables de dimensions qui doivent être liées aux tables de faits. Ces champs constituent, ensemble, les clefs primaires des tables de faits.

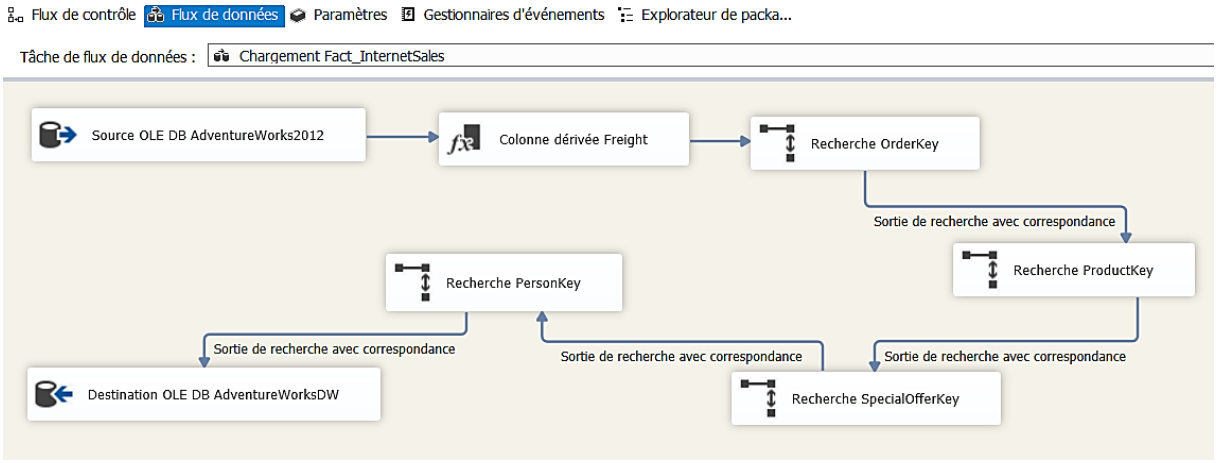


Figure 5.30 : Flux de données - Fact_InternetSales

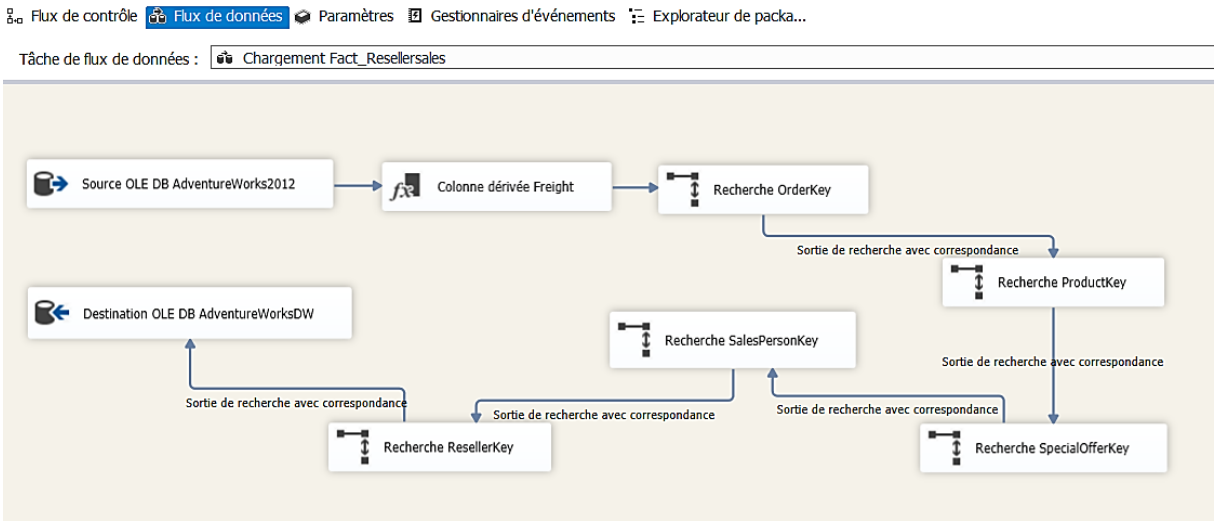


Figure 5.31 : Flux de données - Fact_ResellerSales

Chapitre 6 : Finalisation de l'étude de cas

Ce dernier chapitre, est consacré à la finalisation du développement de l'étude cas qui a été réalisée tout au long de ce mémoire. La première partie du chapitre s'intéresse à la création de la dimension temporelle ainsi qu'à la définition de la sémantique multidimensionnelle des données, avec SQL Server Analysis Services (SSAS). Cet outil fait office de moteur multidimensionnel pour le data mart, et rend possible la formulation de requêtes multidimensionnelles sur celui-ci. La seconde partie du chapitre est quant à elle, dédiée à la présentation rapide de quelques visualisations créées à partir des données du data mart.

6.1 Traitement du data mart avec SQL Server Analysis Services

6.1.1 Mise en place

La première manipulation qui est réalisée avec l'outil, est la création d'une connexion avec le data mart préalablement créé, et dans lequel les données ont été chargées. Il constitue la source de données à partir de laquelle le cube SSAS est créé.

Une vue de cette source de données est ensuite définie. La création de vues de sources de données permet de définir, à partir des schémas des différentes sources, le schéma relationnel des données qui vont effectivement être exploitées lors la définition du cube. Dans le cas présent, puisque la seule source est le data mart des ventes, et que l'objectif est uniquement de définir une structure multidimensionnelle sur ce dernier, l'ensemble de ses tables est incorporé dans la vue.

6.1.2 Génération de la dimension temporelle

Jusqu'à présent, le data mart qui a été créé ne dispose pas d'une dimension temporelle. SSAS permet de remédier à cette situation, en offrant la possibilité de générer une telle dimension, afin qu'elle puisse être utilisée lors de la création de cubes. En plus d'être ajoutée à la vue de source de données, de façon à servir lors de la définition du cube, la table de temps Dim_Time (cf. figure 6.1) est aussi directement ajoutée à la source de données. Cette table est donc directement créée dans le data mart.

L'attribut "Date" est la clef primaire de la table Dim_Time. Pour chacune des deux tables de faits, les attributs "Orderdate", "DueDate" et "ShipDate" sont définis comme des clefs étrangères référençant cette clef primaire. Cette table nouvellement créée est de la sorte véritablement incorporée au data mart.

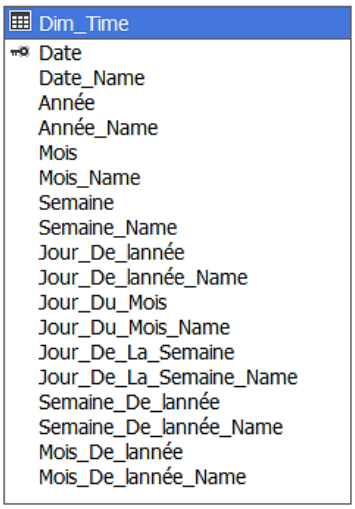


Figure 6.1 : Table Dim_Time

La table rassemble des attributs relatifs à l'année, au mois, à la semaine, ainsi qu'à la date. Deux chemins d'agrégation sont définis par défaut entre la date et l'année (cf. figure 6.2). Le premier passe par le mois et est donc équivalent à la hiérarchie "Calendar" définie sur le schéma conceptuel (cf. figure 2.17). Le second relie pour sa part, les dates aux semaines et les semaines aux années.

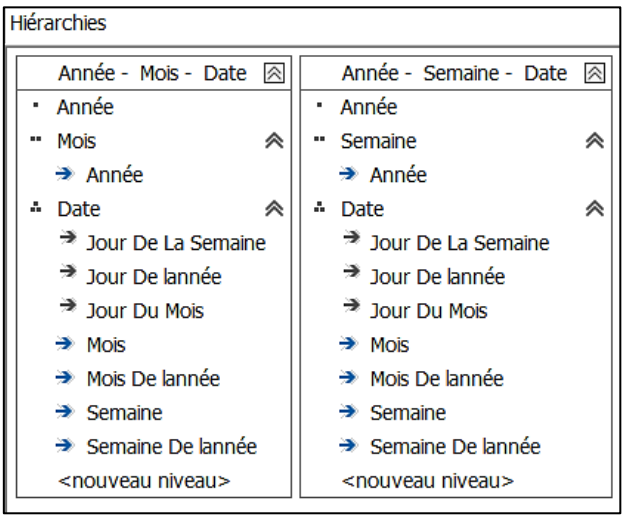


Figure 6.2 : Hiérarchies de la dimension "Time"

Remarque : Les attributs présentant le suffixe "Name" sont des attributs créés par SSAS pour l'affichage des données.

6.1.3 Création du cube

La première étape de la définition du cube est la sélection des groupes de mesures, qui sont en réalité les mesures des deux tables de faits.

La "bridge table" doit elle aussi être déclarée en tant que table de faits, afin de pouvoir être correctement gérée par SSAS. Elle est définie, plus précisément, en tant que table de faits intermédiaire entre les dimensions "Order" et "SalesReason". Cela permet à SSAS de gérer correctement l'agrégation des mesures, malgré la relation many-to-many qui unit les commandes aux raisons d'achat.

L'attribut "Percentage", qui a été calculé lors du processus ETL afin de servir de facteur de distribution dans le cadre de cette hiérarchie non stricte, n'a ici pas à être utilisé. En effet, l'outil est en mesure de résoudre par lui-même le problème de double comptage. Ainsi, lorsque qu'une commande est associée à plusieurs raisons d'achat, la totalité du montant est attribuée à chacune de ces raisons, mais le total général est calculé sans que le montant ne soit compté plusieurs fois. Sur la figure 6.3, on peut constater que la somme des montants associés aux différentes raisons d'achat, est en effet supérieure au total affiché, qui correspond bien au montant total des recettes réalisées grâce aux ventes en ligne. Cette valeur du total montre également que le fait que certaines ventes en ligne ne soient liées à aucune raison d'achat, est aussi correctement géré par l'outil.

Étiquettes de lignes	Receipt - Internet
Marketing	
Television Advertisement	27.475,82 €
Other	
Manufacturer	5.998.122,10 €
Other	248.483,34 €
Price	10.975.842,56 €
Quality	5.549.896,77 €
Review	1.694.881,98 €
Promotion	
On Promotion	6.361.828,95 €
Total général	**29.358.677,22 €**

Figure 6.3 : Recettes des ventes en ligne par raison d'achat

Après les groupes de mesures, ce sont les dimensions qui doivent être définies. Les relations entre les différentes tables dans la vue de source de données, sont ici utilisées par l'outil afin de détecter automatiquement les tables de dimensions associées aux différentes tables de faits. Une fois le cube généré, les différentes dimensions sont éditées afin de définir sur celles-ci, les hiérarchies présentées lors de la phase de design conceptuel (cf. section 2.5).

La mesure dérivée "Receipt" est définie en tant que calcul nommé au niveau de la vue de source de données (cf. figure 6.4), et est ensuite ajoutée comme nouvelle mesure dans les

deux tables de faits du cube. Cette solution est préférée à la définition d'un membre calculé directement dans le cube, pour des raisons de facilité d'agrégation.

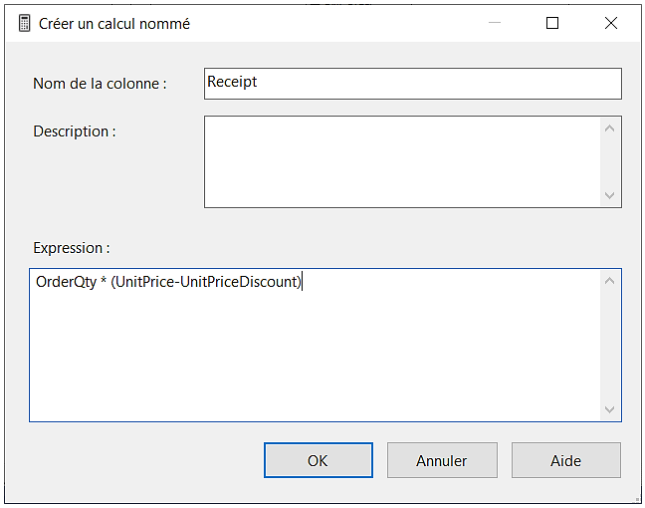


Figure 6.4 : Création de la mesure dérivée "Receipt"

La gestion des mesures non-additives "Unit Price" et "Unit Price discount", nécessite également la réalisation de manipulations au niveau de l'outil. Ce dernier ne donne en effet pas la possibilité d'agréger des mesures avec la moyenne. Ces deux mesures sont donc définies comme additives pour les deux tables de faits, et des membres calculés sont créés afin de reproduire le comportement d'une moyenne (cf. figure 6.5). L'agrégation par la somme est transformée en agrégation par la moyenne, au moyen d'une division par le nombre d'instances.

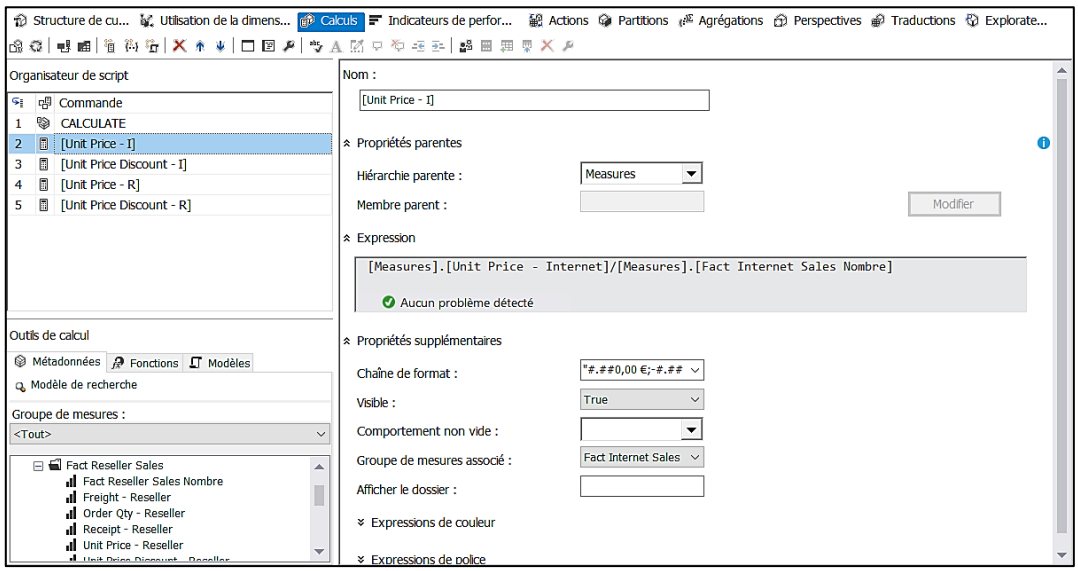


Figure 6.5 : Création du membre calculé "Unit Price"

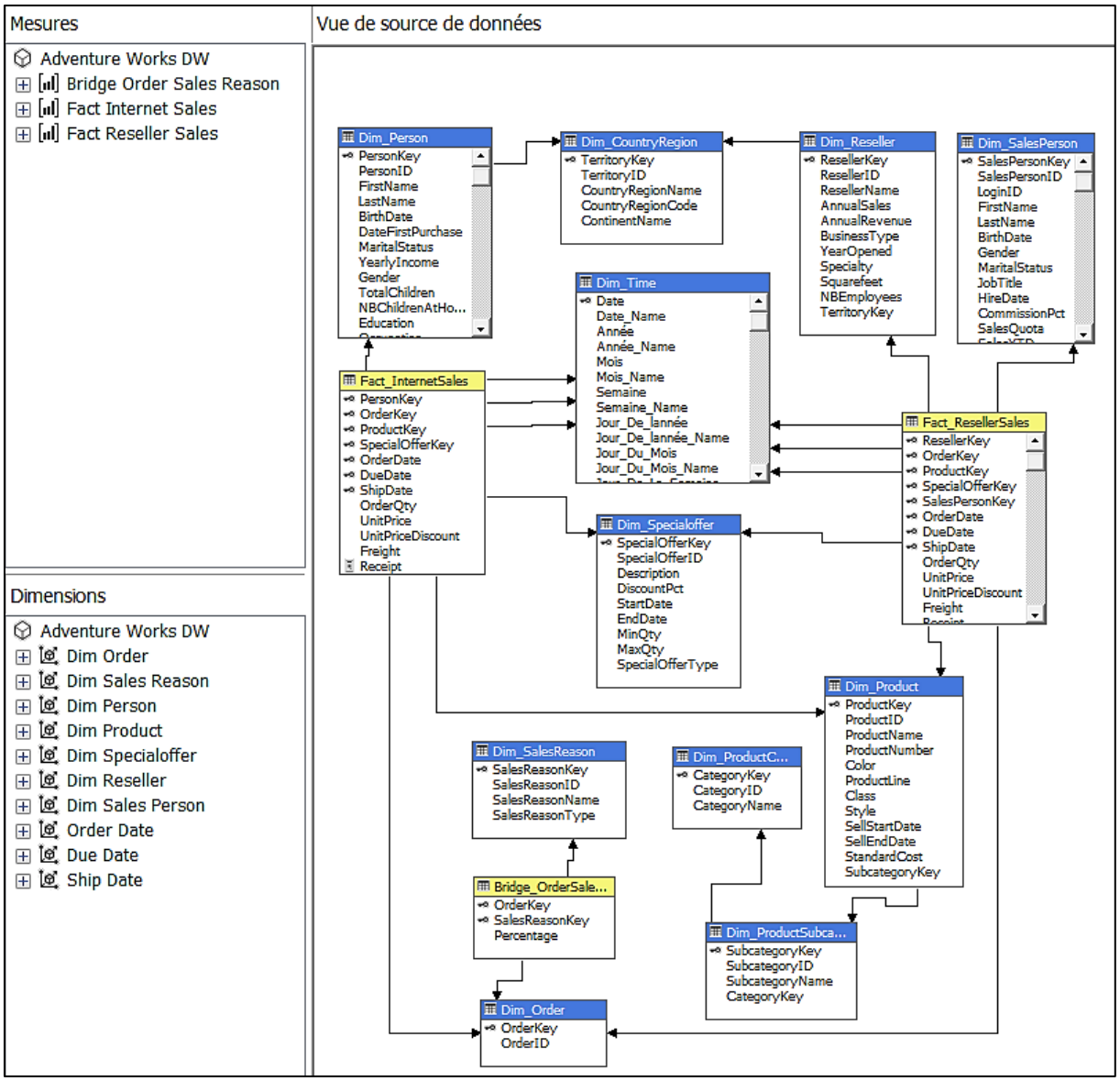


Figure 6.6 : Cube SSAS du data mart des ventes

6.2 Quelques exemples de visualisations

Maintenant que le cube a été créé, du reporting peut être réalisé sur les données qu'il contient. Toutefois, le reporting étant en dehors du cadre de ce mémoire, seules quelques visualisations sont fournies à titre illustratif.

Les quelques figures qui suivent sont des visualisations réalisées avec les données du data mart grâce aux outils Microsoft Excel et SQL Server Reporting Services (SSRS). Alors que SSAS offre directement la possibilité d'explorer le contenu d'un cube avec Excel, SSRS nécessite la création d'une connexion avec le cube et le passage par l'éditeur de requêtes MDX.

Receipt - Reseller	Étiquettes de colonnes				
Étiquettes de lignes	Calendar 2005	Calendar 2006	Calendar 2007	Calendar 2008	Total général
Europe					
France		860.884,67 €	2.406.047,63 €	1.380.371,28 €	4.647.303,59 €
Germany			1.144.807,18 €	906.620,18 €	2.051.427,36 €
United Kingdom		844.231,18 €	2.186.685,93 €	1.280.062,39 €	4.310.979,50 €
North America					
Canada	1.514.283,05 €	4.861.916,51 €	5.691.007,50 €	2.395.711,42 €	14.462.918,48 €
Central	951.798,89 €	2.635.685,66 €	3.016.214,27 €	1.328.990,14 €	7.932.688,96 €
Northeast	568.551,97 €	2.453.728,90 €	2.876.086,75 €	1.058.137,91 €	6.956.505,53 €
Northwest	1.690.236,55 €	3.502.528,46 €	4.687.225,43 €	2.642.778,03 €	12.522.768,48 €
Southeast	1.450.812,02 €	2.838.481,60 €	2.444.492,77 €	1.174.376,63 €	7.908.163,03 €
Southwest	1.894.063,98 €	6.330.850,97 €	7.188.233,79 €	3.184.454,64 €	18.597.603,38 €
Pacific					
Australia			875.575,62 €	747.235,07 €	1.622.810,68 €
Total général	**8.069.746,47 €**	**24.328.307,96 €**	**32.516.376,87 €**	**16.098.737,69 €**	**81.013.168,98 €**

Figure 6.7 : Recettes des ventes hors ligne par zone géographique et par année

La première visualisation est un tableau croisé dynamique réalisé avec Excel (cf. figure 6.7). Il présente, par année, les recettes réalisées grâce aux ventes auprès de revendeurs, dans les différentes zones géographiques où l'entreprise est active.

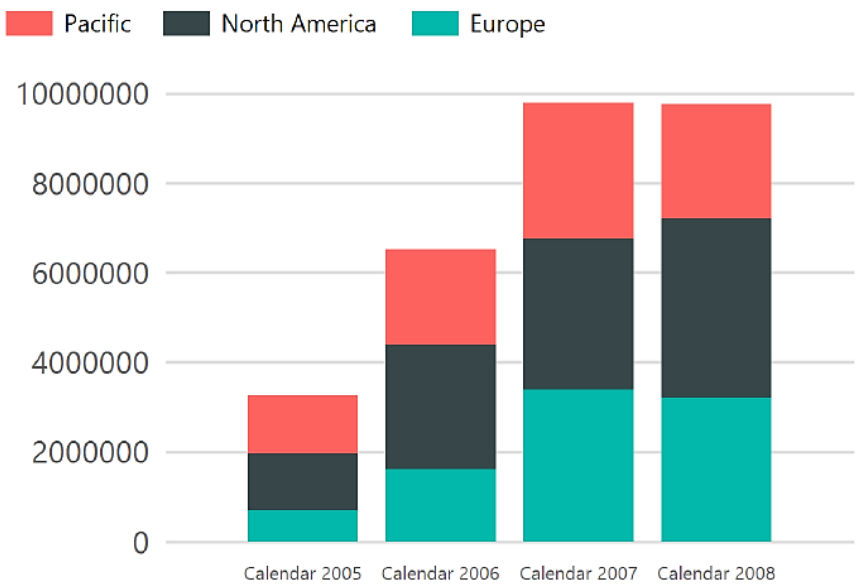


Figure 6.8 : Recettes des ventes en ligne par continent et par année

La deuxième visualisation est un "stacked bar chart" réalisé avec SSRS (cf. figure 6.8). Il donne un aperçu de l'évolution d'année en année, des recettes des ventes en lignes, ainsi que de l'importance des différents continents dans ces recettes.

Enfin, le troisième et dernier exemple de visualisation est un "pie chart" lui aussi créé avec SSRS (cf. figure 6.9). Il renseigne sur la répartition des recettes réalisées en 2007 grâce aux ventes en ligne, selon du niveau d'éducation des acheteurs.

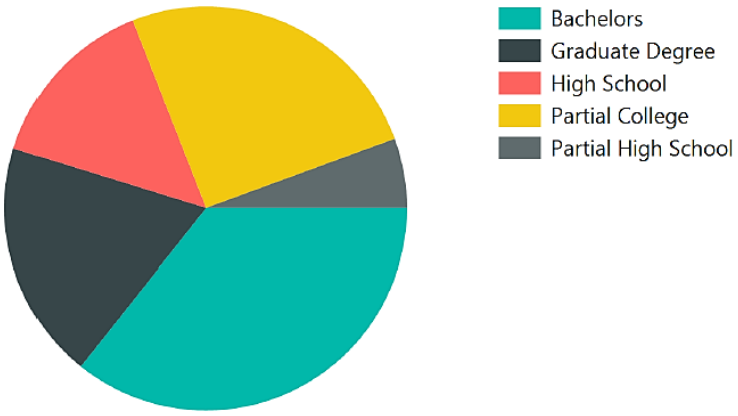


Figure 6.9 : Répartition des recettes des ventes en ligne en 2007 selon le niveau d'éducation des acheteurs

Conclusion

L'objectif de ce travail a été l'exposition d'une méthodologie complète de développement d'un data warehouse, combinée à la confrontation de celle-ci à la pratique, grâce au développement d'une étude cas.

Après un premier chapitre de mise en contexte, tant sur le plan théorique que concernant l'étude de cas qui est développées dans le reste du mémoire, le deuxième chapitre a été consacré à la phase de design conceptuel. Il a débuté avec la mise en évidence de la tendance à négliger cette étape, malgré l'apport considérable de sa réalisation au reste du processus de design. Ce chapitre a aussi permis de présenter différentes manières d'aborder la spécification des exigences pour ce type de systèmes, et leur impact sur le reste de l'étape conceptuelle. Enfin, la notation Multidim a été présentée, et a ensuite été utilisée pour la réalisation du schéma conceptuel du data mart de l'étude de cas.

Le troisième chapitre a ensuite été consacré à la phase de design logique. Les modèles ROLAP, MOLAP, et HOLAP ont été rapidement décrits, et leurs avantages et inconvénients respectifs ont été soulignés. Une attention particulière a ensuite été accordée au modèle ROLAP, au travers de la présentation du Star schema, du Snowflake schema, et d'autres variantes de ce type d'architectures. Le chapitre s'est ensuite terminé avec la traduction du schéma conceptuel de l'étape précédente, en schéma logique.

En ce qui concerne le chapitre 4, il y a été question du design physique des data warehouses, et les trois grandes méthodes d'optimisation des performances de ces systèmes ont été discutées.

Le chapitre 5 a quant à lui été dédié à la phase d'ETL. Après avoir défini le concept de processus ETL et décrit ses différentes étapes, la question de la modélisation de ce type de processus et le problème de l'absence d'un modèle standard pour cette dernière, ont été abordés. La suite du chapitre s'est vu consacrer à la présentation d'une approche pour le design conceptuel des processus ETL, ainsi qu'à son application dans le cadre de l'étude de cas. Le chapitre s'est finalement terminé avec la mise en place du processus ETL du data mart de l'étude de cas, avec l'outil SQL Server Integration Services.

Enfin, le dernier chapitre a consisté en la finalisation de l'étude de cas, avec la définition d'un cube SSAS à partir du data mart préalablement créé et chargé. Quelques visualisations ont également été réalisées, à titre d'exemple d'utilisation du data mart et de ses données.

La principale conclusion qui peut être tirée de ce mémoire, est sans doute qu'il est crucial de suivre une méthodologie complète dans le cadre d'un projet de data warehousing, et qu'aucune des étapes ayant été abordées, ne peut être négligée. Il peut être tentant pour des responsables de projets, de tenter d'accélérer le processus en ignorant l'une ou l'autre étape, notamment lorsqu'il n'existe pas de standard méthodologique ou en matière de notation pour l'étape en question. Or, chacune des phases du processus est importante, et ne pas apporter le soin nécessaire à certaines d'entre elles, peut grandement compromettre les chances de réussite du projet.

En plus de réduire les efforts à produire lors des phases suivantes, l'étape de design conceptuel permet de faire en sorte que le système soit adapté aux besoins d'analyse des utilisateurs. Négliger les phases de design logique et physique peut pour sa part, aboutir à un système défaillant ou insuffisamment performant. Enfin, un design minutieux du processus ETL garantit que les analyses seront réalisées à partir de données cohérentes et correctes.

En ce qui concerne les limitations dont souffre ce mémoire, la plus importante d'entre elles est inhérente à l'objectif de ce dernier. Étant donné qu'une méthodologie complète a été présentée, il n'a été possible de s'attarder en détails sur aucune des étapes. Il a fallu faire des choix, et se montrer bref par endroits. En réalité, alors qu'un chapitre a été consacré à chacune des étapes du processus, en vue de pouvoir exposer celui-ci dans son ensemble, chacune d'elles aurait pu faire l'objet d'un mémoire entier. Ainsi, une revue approfondie de la littérature concernant l'étape de design conceptuel, avec une comparaison des différents modèles proposés, aurait pu être un sujet pertinent. De même, consacrer un mémoire entier à la phase de design logique, aurait par exemple permis d'analyser avec beaucoup plus de détails les modèles ROLAP, MOLAP et HOLAP.

Cette limitation se fait particulièrement sentir au niveau du chapitre 4, consacré à la phase de design physique. Pour ce chapitre, la décision a été prise de se contenter d'un aperçu théorique des trois grandes méthodes d'optimisation que sont les vues matérialisées, les index, et le partitionnement, et de laisser les outils utilisés gérer par défaut, l'optimisation du data mart de l'étude de cas. Si plus de temps et d'espace avait pu être consacrés à cette partie du processus, il aurait par exemple été pertinent, de s'intéresser en détails à la question de la sélection des vues matérialisées et des index, ou encore de tester différents types d'index ou différentes méthodes de partitionnement, afin de comparer les gains de performance obtenus.

Bibliographie

Articles et ouvrages

- Ariyachandra, T. et Watson, H.J. (2006). Which Data Warehouse Architecture Is Most Successful? *Business Intelligence Journal*, 11 (1), pp. 4-6.
- Bouzeghoub, M. et Kedad, Z. (2000). A Quality-Based Framework for Physical Data Warehouse Design. Dans: Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses. Stockholm, Suède.
- Colliat, G. (1996). OLAP, Relational, and Multidimensional Database Systems. *SIGMOD Record*, 25 (3), pp. 64-69.
- Di Tria, F., Lefons, E. et Tangorra, F. (2017). Cost-benefit analysis of data warehouse design methodologies. *Information Systems*, 63, pp. 47-62.
- El-Sappagh, S., Hendawi, A. et El Bastawissy A. (2011). A proposed model for data warehouse ETL processes. *Journal of King Saud University – Computer and Information Sciences*, 23, pp. 91-104.
- Franconi, E. et Sattler, U. (1999). A Data Warehouse Conceptual Data Model for Multidimensional Aggregation. Dans: Proceedings of the 1st International Workshop on Design and Management of Data Warehouses. Heidelberg, Allemagne.
- Golfarelli, M., Maio, D. et Rizzi, S. (1998). Conceptual Design of Data Warehouses from E/R Schemes. Dans: Proceedings of the 31st Hawaii International Conference on System Sciences. Kohala Coast, USA: IEEE.
- Golfarelli, M. et Rizzi, S. (2009). Data Warehouse Design: Modern principles and Methodologies. New York: McGraw-Hill Education.
- Gupta, H., Harinarayan, V., Rajaraman, A. et Ullman, J. (1997). Index selection for OLAP. Dans: Proceedings of the 13th International Conference on Data Engineering. Birmingham, Royaume Uni, pp. 208-219.
- Hani Zulkifli Abai, N., Yahaya, J. H. et Deraman, A. (2013). User Requirement Analysis in Data Warehouse Design : A Review. *Procedia Technology*, 11, pp. 801-806.
- Kimball, R. et Ross, M. (2002). The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. New York: Wiley.
- Kimball, R., Ross, M., Thornthwaite, W., Mundy, J. et Becker, B. (2008). The Data Warehouse Lifecycle Toolkit. Indianapolis: Wiley.
- Linden, I. (2018). The Multidimensional Model, notes de cours, Business Intelligence [EDASM101], Université de Namur, Février 2018.
- Luján-Mora, S., Trujillo, J. et Song, I. (2006). A UML profile for multidimensional modeling in data warehouses. *Data & Knowledge Engineering*, 59, pp. 725-769.

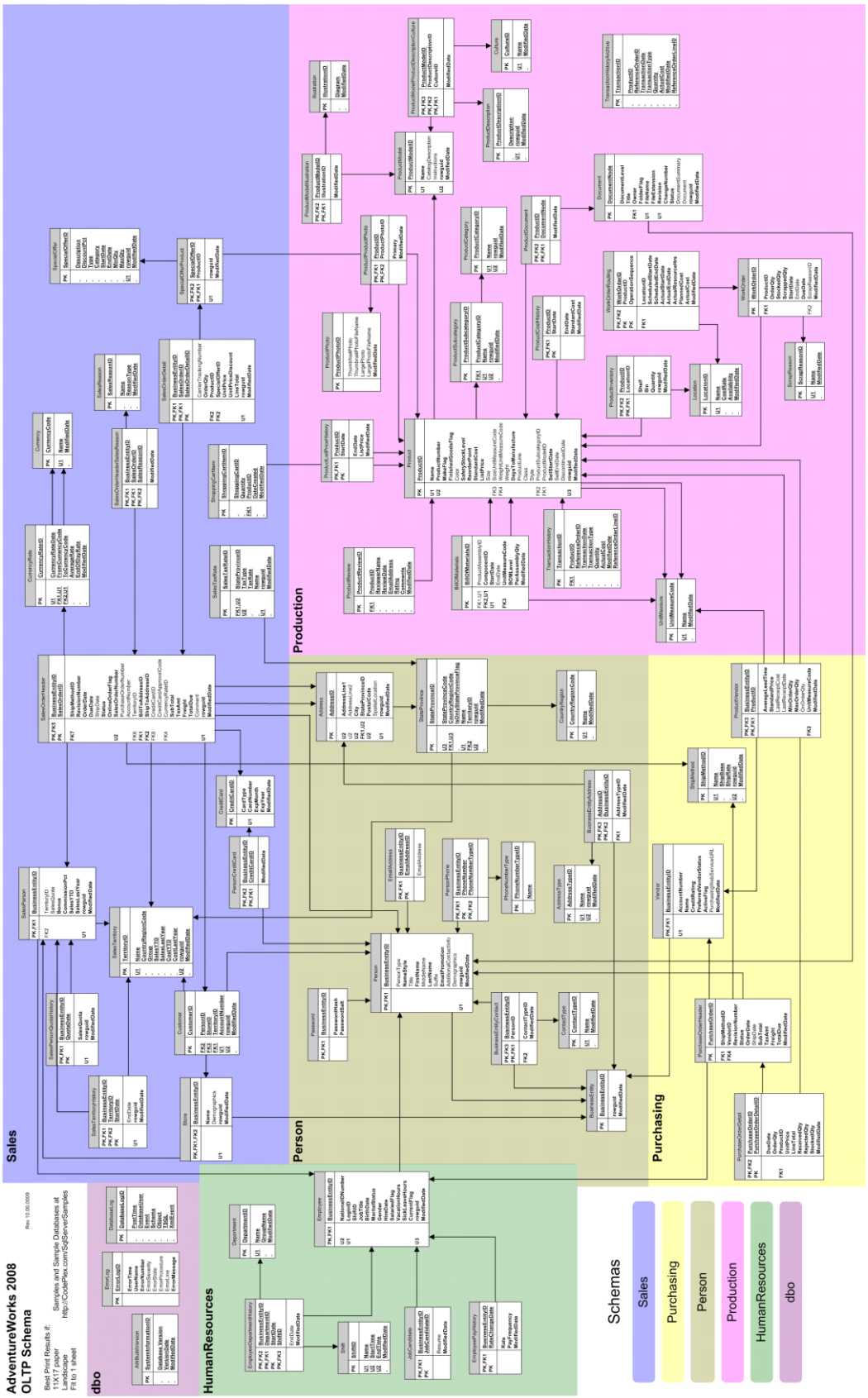
- Martyn, T. (2004). Reconsidering Multi-Dimensional Schemas. *SIGMOD Record*, 33 (1), pp. 83-88.
- Sonal, S. et Rajni, J. (2014). Modeling ETL Process in Data warehouse: An Exploratory Study. Dans: Proceedings of the 4th International Conference on Advanced Computing & Communication Technologies. Rohtak, India, IEEE, pp. 271-276.
- Trujillo, J. et Luján-Mora, S. (2003). A UML Based Approach for Modeling ETL Processes in Data warehouses. Dans: Proceedings of the 22nd International Conference on Conceptual Modeling. Chicago, USA, Springer, pp. 307-320.
- Tryfona, N., Busborg, F. et Borch Christiansen, J. G. (1999). starER: A Conceptual Model for Data Warehouse Design. Dans: Proceedings of the 2nd ACM international Workshop on Data warehousing and OLAP. Kansas City, USA, pp. 3-8.
- Vaisman, A. et Zimányi, E. (2014). Data Warehouse Systems: Design and Implementation. Berlin: Springer-Verlag.

Sites Internet

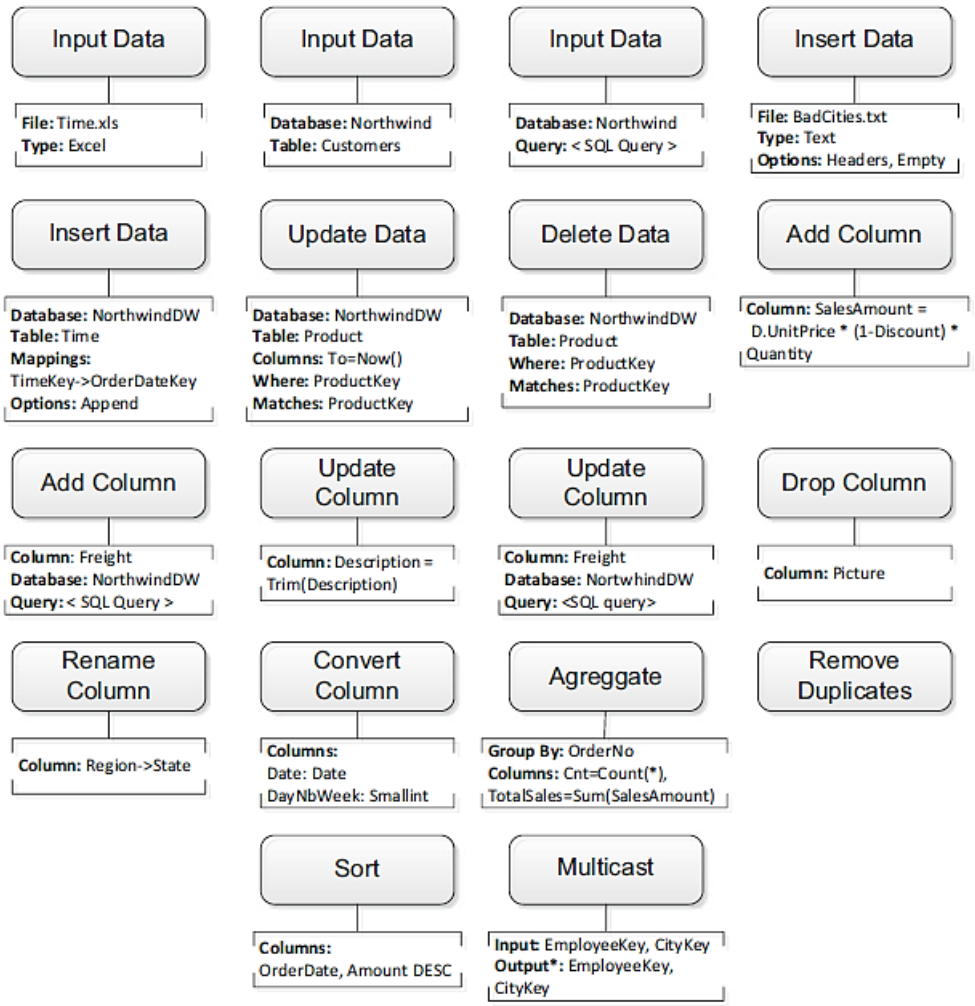
- Datawarehouse4u.info (2019). Data Warehouse Schema Architecture – star schema | Datawarehouse4u.info [Online]. Disponible à : https://www.datawarehouse4u.info/Data-warehouse-schema-architecture-star-schema.html [Consulté le 08 MAR 2019].
- Microsoft.com (2010). AdventureWorks Data Dictionary | Microsoft Docs [Online]. Disponible à : https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008/ms124438(v=sql.100)
- Sqldatadictionary.com (2016). AdventureWorks2014: AdventureWorks 2014 Sample OLTP Database [Online]. Disponible à : https://www.sqldatadictionary.com/AdventureWorks2014.pdf

Annexes

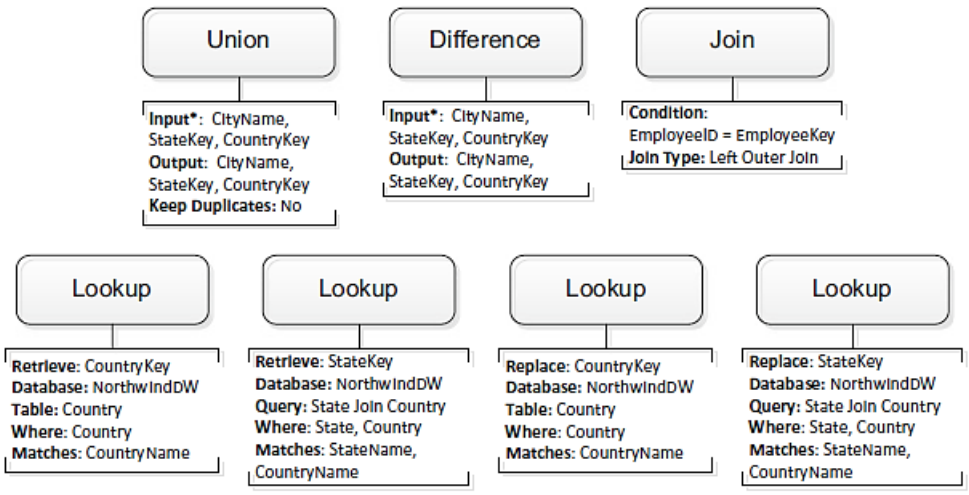
Annexe 1 : Schéma de données de l'ERP d'Adventureworks



Annexe 2 : BPMN - Liste des Data Tasks (Vaisman et Zimányi, 2014)



Unary data tasks



N-ary data tasks